Fachhochschule Köln
University of Applied Sciences Cologne
Abteilung Gummersbach
Fachbereich Informatik

Diplomarbeit
(Drei-Monats-Arbeit)
zur Erlangung
des Diplomgrades
Diplom-Informatiker (FH)
in der Fachrichtung Informatik

„Frameworks bei der Entwicklung von Web-Applikationen am Beispiel von Jakarta Struts und Oracle MVC Framework“

Erstprüfer:	Fr. Prof. Dr. Rer. Nat. Heide Faeskorn-Woyke
Zweitprüfer:	Hr. Prof. Dr. Dipl.-Inform. Frank Victor
vorgelegt am:	19. September 2002
von cand.:	Guido Niebel
aus:	Zülpicher Str. 351
	50937 Köln
Tel.-Nr.:	0221 / 61 88 35
E-Mail:	wi242@gm.fh-koeln.de
Matr.-Nr.:	11022415

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ASP: Active Server Pages

BC4J: Business Components for Java

CGI: Common Gateway Interface

CTS: Compatibility Test Suite

DOM: Document Object Modell

EIS: Enterprise Information System

EJB: Enterprise Java Beans

GUI: Graphical User Interface

HIS: Hochschul Informations System

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IIS: Internet Information Server

J2EE: Java 2 Enterprise Edition

JCP : Java Community Process

JDBC: Java Database Connectivity

JSP: Java Server Pages

MVC: Modell View Controller

PDF: Portable Document Format

PHP: PHP Hypertext Preprocessor

SQL: Structured Query Language

UIX: User Interface XML

UML: Unified Modelling Language

URL: Uniform Resource Locator

VM: Virtual Machine

WML: Wireless Markup Language

XML: Extensible Markup Language Compiler

XMLC: Extended Markup Language Compiler

Kapitel 1

Einleitung

Es ist eine Selbstverständlichkeit geworden, über das Internet Flüge zu buchen, Überweisungen auszuführen oder Produkte zu kaufen. All dies führt man mit einem Browser durch, so dass oft der Eindruck entsteht, der Browser sei die fürs Internet entscheidende Applikation. Dass dieser häufig lediglich zum Anzeigen von Web-Seiten und der Verwendung von Formularen in diesen dient, ja dass die gesamte Applikationslogik bei den oben beschriebenen Vorgängen meist gar nicht im Browser, sondern auf Web-Servern oder tief in den Unternehmen verwendeten Application-Servern hinterlegt ist, bemerkt man hierbei nicht.

Um diese Komplexität einerseits zu verbergen, sie andererseits aber in einer leicht verständlichen und homogenen Umgebung zur Verfügung zu stellen, ist mit dem Internet eine neue Applikationsart entstanden: die Web-Applikation.

Von dieser handelt Kapitel 2, in dem zunächst die charakteristischen Eigenschaften von Web-Applikationen in einer Definition zusammengefasst werden. Aus diesen Eigenschaften resultieren Anforderungen, die von Web-Applikationen und den ihnen zugrunde liegenden Techniken erfüllt werden müssen. Nach deren Erörterung werden die Vorteile von Web-Applikationen erläutert, aus denen erkennbar ist, warum sie mittlerweile so häufig verwendet werden. Da sie aber auch einigen Einschränkungen unterliegen, werden abschließend ihre Nachteile aufgeführt.

Zur Entwicklung dieser Web-Applikationen sind in den letzten Jahren zahlreiche Techniken entstanden. Einige von ihnen können innerhalb des Browsers verwendet werden, andere werden innerhalb der Web-Server eingesetzt. Zum Zugriff auf Unternehmensdaten und -anwendungen dienen Application-Server und Datenbanken. Kapitel 3 soll einen Überblick über diese Techniken geben und ihren Einfluss auf die Entwicklung von Web-Applikationen aufzeigen.

Nach der Vielzahl der zur Entwicklung von Web-Applikationen vorgestellten Techniken könnte man den Eindruck gewinnen, diese alleine wären ausreichend für eine sinnvolle Entwicklung. Aber wie jede Technik und insbesondere jede Technik in der EDV, lassen sich auch diese falsch oder ineffizient anwenden. Zu ihrer sinnvollen Nutzung sind feste Rahmen notwendig, in denen sich Entwickler bewegen können, um Web-Applikationen zu erstellen, die auf Erfahrungen und Vorgehensweisen bei der erfolgreichen Anwendung dieser Techniken beruhen: die Frameworks. Kapitel 4 beschreibt zunächst ihre Grundlagen und betrachtet dann anhand von typischen Architekturen ihre Grundzüge in Bezug auf Web-Applikationen. Anschließend werden eine Vielzahl von Frameworks auf Grundlage von J2EE vorgestellt, die in ihrer Gesamtheit zeigen sollen, auf welch immensen Erfahrungsschatz bei der Erstellung von sehr unterschiedlich gearteten Web-Applikationen mit Hilfe von Frameworks zugegriffen werden kann.

In den beiden folgenden Kapiteln 5 und 6 sollen mit Struts und dem Oracle MVC Framework zwei dieser Frameworks detaillierter betrachtet werden. Sie besitzen ein vergleichbares Anwendungsgebiet, setzen bei der Umsetzung der ihnen zugrunde liegenden Modell-View-Controller Architektur aber unterschiedliche Schwerpunkte. So soll ihre Be-

schreibung auch unterschiedliche Aspekte und Auswirkungen bei der Entwicklung von Web-Applikationen unter Verwendung von Frameworks aufzeigen. Um das Arbeiten mit ihnen besser verständlich machen zu können, wird in den Kapiteln die Entwicklung einer Web-Applikation im Umfeld der Fachhochschule beschrieben, deren Umsetzung mit Struts auch auf der beigefügten CD enthalten ist.

Abgeschlossen wird die Arbeit mit Kapitel 7, in dem die Ergebnisse der Arbeit in einem Fazit zusammengefasst werden.

Kapitel 2

Web-Applikationen

Es erfolgt zunächst eine Definition des Begriffes *Web-Applikation*, da er für unterschiedlichste Applikations-Arten verwendet wird.

Von dieser Definition ausgehend, werden anschließend Eigenschaften und die hieraus resultierenden Anforderungen an Web-Applikationen beschrieben.

Aufgrund dieser Eigenschaften besitzen Web-Applikationen einige Vorteile, die kurz aufgeführt werden sollen.

Kein Licht ohne Schatten und so wird abschließend auf die Nachteile der Web-Applikationen eingegangen.

2.1 Definition

Der Begriff *Web-Applikation* wird allgemein für unterschiedliche Anwendungen, die eine Beziehung zum Internet haben, verwendet. Zu diesen gehören E-Mail-Clients, Programme zum Online-Banking oder auch Web-Services. Da in dieser Arbeit aber nur ein Teil dieser Anwendungen von Interesse ist, soll zunächst eine Definition dessen angeben werden, was im weiteren als Web-Applikation gemeint ist. Ziel der Definition ist es somit, die im weiteren behandelten Web-Applikationen durch eine möglichst genaue Beschreibung ihrer Charakteristik von anderen Arten abzugrenzen.

> Eine Web-Applikation ist eine verteilte Mehrbenutzer-Anwendung. Ihre Aufgabe ist es, den Benutzern Informationen und Prozesse, die von Servern zur Verfügung gestellt werden, innerhalb einer einheitlichen Oberfläche in einem Browser über das World Wide Web zur Verfügung zu stellen.

Auf Grundlage dieser Definition lassen sich nun Eigenschaften von Web-Applikationen beschreiben. Falls sich durch diese Anforderungen an die Web-Applikation oder an ihre Entwicklung ergeben, werden sie ebenfalls kurz erläutert.

2.2 Eigenschaften und Anforderungen

Verteilte Anwendung

Web-Applikationen können Komponenten sowohl im Browser als auch auf Servern verwenden. Die Kommunikation bei diesen verteilten Applikationen wird hierbei über das Internet, speziell über das HTTP-Protokoll, geführt.

Sicherheit

Im Unterschied zu verteilten Anwendungen, die unternehmensintern ablaufen, steht somit zwischen Benutzern und Server das vom Sicherheitsstandpunkt vertrauensunwürdige Internet. Es ist somit wichtig bei vertrauenswürdigen Vorgängen, z.B. Überweisungen oder dem Zugriff auf geschützte Daten, die Kommunikation durch Verschlüsselung abzusichern.

Verfügbarkeit

Da Web-Applikationen nur in Verbindung mit einem Server lauffähig sind, ist dessen Verfügbarkeit ebenfalls von großer Bedeutung. Hierbei ist dafür zu sorgen, dass beim Ausfall eines Servers weitere Server bereitstehen, um dies abzufangen. Ist Verfügbarkeit eine unverzichtbare Anforderung an eine Web-Applikation, so schränkt dies die verwendbare Technik eventuell ein.

Verteilte Applikationslogik

Durch die Verteilung der Applikation auf räumlich getrennte Bestandteile müssen für die Erstellung der Applikationslogik neue Konzepte verwendet werden. Erschwerend kommt hinzu, das Benutzer in den Browsern Aktionen ausführen können, die Einfluss auf die

als nächstes auszuführende Applikationslogik haben, jedoch von der Applikation lediglich festgestellt und nicht kontrolliert werden können. Beispiele hierfür sind das Öffnen mehrerer Browser-Fenster oder das Zurückspringen innerhalb des Browsers. Dieses Benutzerverhalten muss bei der Erstellung der Applikation unbedingt berücksichtigt werden.

Mehrbenutzer-Anwendung

Eine Web-Applikation wäre nicht brauchbar, wenn nicht zeitgleich mehrere Benutzer auf sie zugreifen könnten. Hieraus resultieren die folgenden Anforderungen:

Verwalten von Benutzerinformationen

Bei personalisierten Web-Applikationen müssen die Benutzer serverseitig erkannt und verwaltet werden. Sollen mehrere solcher Applikationen zusammen genutzt werden, empfiehlt es sich, diese Benutzerinformationen an einem zentralen Punkt zu verwalten. Hierdurch kann ein ansonsten notwendiges mehrmaliges Einloggen des Benutzers verhindert werden.

Rechteverwaltung

Unterschiedliche Benutzer einer Web-Applikation haben unterschiedliche Rechte für den Zugriff auf Dokumente und die Verwendung von Geschäftsprozessen. Eine reine Benutzerverwaltung kann dem nicht Rechnung tragen. In diesen Fällen ist eine Rechteverwaltung notwendig, die, da sie weitreichenden Einfluss auf die Erstellung der Applikation hat, von Anfang an in deren Entwicklung berücksichtigt werden sollte.

Skalierbarkeit

Oft lässt sich beim Erstellen einer Web-Applikation nicht einschätzen, wie viele Benutzer letztendlich auf sie zugreifen werden. Sind hier große Schwankungen möglich, so müssen die verwendete Technik und die mit ihr erstellte Web-Applikation skalierbar sein. Durch eine suboptimale Architektur lässt sich zudem jegliche Web-Applikation unskalierbar machen, weshalb dieser Aspekt bei der Applikationsentwicklung ausreichend berücksichtigt werden sollte.

Bereitstellung von Informationen und Prozessen

Über Web-Applikationen werden nicht nur statische Web-Seiten abgerufen, sie dienen auch zum Zugriff auf Datenbanken sowie dem Ausführen von Geschäftsprozessen. Während zum Beispiel bei Online-Zeitungen die Informationsübermittlung im Mittelpunkt steht, sind es bei Brokerage-Systemen die Prozesse, bei Web-Shops ist häufig beides von Bedeutung.

Einbindung von Unternehmensanwendungen

Geschäftsprozesse werden außerhalb des Unternehmens ausgelöst, müssen jedoch auf unternehmensinternen Systemen ausgeführt werden. Dies erzwingt die Anbindung unterschiedlichster Unternehmensanwendungen an die Web-Applikation. In diesen Fällen ist eine äußerst flexible Plattform zur Erstellung der Web-Applikation notwendig.

Workflow

Geschäftsprozesse folgen einer bestimmten Logik, die innerhalb der Web-Applikation implementiert und dem Benutzer angezeigt werden muss. Die Web-Applikation hat zudem die Aufgabe, den korrekten Ablauf dieses Workflows, z.B. das Aufgeben einer Bestellung, zu gewährleisten. Keine der verwendbaren Plattformen erfüllt von Grund auf diese Anforderung.

Einheitliche Oberfläche

Web-Applikationen präsentieren den Benutzern Informationen und Prozesse innerhalb einer einheitlich gestalteten Oberfläche. So kann sichergestellt werden, dass die Applikation auch als zusammenhängend erkannt wird.

Ergonomie

Wie bei allen Benutzeroberflächen ist hierbei auf die Ergonomie der Anwendung zu achten. Erschwerend kommt hinzu, dass die für die Benutzeroberfläche verwendeten Browser im Vergleich zu Desktop-Applikationen nur eine geringe Anzahl an Bedienelementen zur Verfügung stellen.

Trennung von Design und Programmierung

Da sowohl Design als auch Programmierung einer Web-Applikation sehr umfangreich sind, empfiehlt es sich, beide Aufgabenbereiche stark voneinander zu trennen. Hierdurch können in den jeweiligen Bereichen spezialisierte Entwickler getrennt voneinander arbeiten. Diese Trennung ist jedoch nicht mit jeder Technik realisierbar und wird je nach Umsetzung unterschiedlich stark unterstützt.

Benutzung über einen Browser

In dieser Arbeit werden lediglich Applikationen berücksichtigt, die über einen Web-Browser bedient werden, also insbesondere keine Desktop-Anwendungen wie Email-Clients oder FTP-Programme, die durchaus auch als Web-Applikationen bezeichnet werden. Auch werden keine Applikationen berücksichtigt, die gänzlich ohne Benutzeroberfläche auskommen, wie dies zum Beispiel bei Web-Services der Fall ist.

Benutzung über das World Wide Web

Innerhalb eines Browsers können unterschiedliche Internet-Protokolle, wie HTTP oder FTP, genutzt werden. Auch hier ist eine Einschränkung sinnvoll und es sind lediglich Applikationen gemeint, die auf Grundlage von HTTP arbeiten.

2.3 Vorteile

Nachdem nun die in dieser Arbeit behandelten Web-Applikationen ausreichend definiert wurden, sollen zunächst ihre Vorteile aufgeführt werden.

Ein entscheidender Vorteil von Web-Applikationen gegenüber anderen Applikationsarten liegt in ihrer Verfügbarkeit. Sie müssen nicht an Kunden ausgeliefert oder auf Arbeitsplatzrechnern installiert werden, sondern sind mit der Installation auf einem Web-Server sofort von jedem internetfähigen Rechner aus nutzbar. Häufig ist ein Web-Server ohnehin schon vorhanden, so dass es auch zu einer Kostenersparnis kommt.

Durch ihr Vorhandensein auf einem Server sind Web-Applikationen zudem zentral konfigurierbar und Updates werden sofort von ihnen verwendet. Zusätzlich können Informationen über das Benutzerverhalten gesammelt werden, was z.B. bei Web-Shops alleine schon einen Mehrwert darstellen kann.

Die Einfachheit der Browser, deren Benutzung den Anwendern zudem vertraut ist, erlaubt es, Web-Applikationen ohne umfangreiche Schulung zur Verfügung zu stellen. Zudem werden hierdurch betriebssystemübergreifende Anwendungen erstellt, ein Punkt, der mit herkömmlichen Applikationen nur schwer zu erfüllen ist.

Und schließlich lassen sich mit Web-Applikationen Anwendungen realisieren, die sonst nicht machbar wären, wie z.B. Unternehmensportale, Web-Shops oder Online-Zeitungen.

2.4 Nachteile

Ein Nachteil ist sicherlich der momentan noch eingeschränkte Einsatzbereich der Web-Applikationen. Aufgrund der im Vergleich zu Desktop-Applikationen funktionsarmen Browser lassen sich rechen- oder grafikintensive Anwendungen wie Tabellenkalkulationen oder CAD-Anwendungen nicht mit ihnen realisieren.

Negativ wirkt sich hier auch die Notwendigkeit aus, große Datenbestände vor der Bearbeitung erst über das Internet transportieren zu müssen. Um dies zu verhindern, werden auch bei heutigen Applikationen Datenbestände meist serverseitig verwaltet, was hier zu Engpässen führen kann.

Die Erstellung einer Web-Applikation ist zudem ein äusserst komplexer Vorgang. Hierbei muss eine Applikation sowohl mit unterschiedlichen Techniken erstellt werden, es muss aber auch die Arbeit unterschiedlicher Entwicklergruppen, wie Web-Designern, Anwendungsentwicklern und Systemadministratoren, koordiniert werden.

Und schließlich existieren für die Entwicklung von Web-Applikationen zwar eine Reihe an Best-Practices, von denen einige in dieser Arbeit vorgestellt werden, jedoch gibt es keine festen Vorgehensweisen, die eine erfolgreiche Entwicklung von vorne herein garantieren. Zusätzlich erschwert wird dies durch ständig neue Techniken, wie zuletzt z.B. Web-Services, die zur Umsetzung genutzt werden können.

Kapitel 3

Techniken für Web-Applikationen

Nach der Beschreibung der Web-Applikationen soll nun auf die Techniken eingegangen werden, mit denen sie erstellt werden können.

Zunächst werden die Web-Browser und die in ihnen clientseitig anwendbaren Techniken vorgestellt.

Anschließend wird beschrieben, welche Techniken innerhalb der Web-Server verwendet werden können.

Danach werden die Application-Servern betrachtet, welche die Einsatzmöglichkeiten von Web-Applikationen auf die Unternehmensanwendungen ausweiten.

Das Kapitel wird abgeschlossen mit den wichtigsten zu beachtenden Aspekten bei der Benutzung von Datenbanken.

3.1 Web Browser

Web-Browser dienen überwiegend zur Präsentation der Web-Applikation. Es existieren aber auch einige Techniken, mit denen Teile der Applikation innerhalb des Browsers ausgeführt werden können.

3.1.1 Übersicht

Mitte der 90er Jahre besaß der Netscape Navigator noch einen Marktanteil von über 90%. In den Jahren danach, theatralisch auch gerne als die Zeit der Browser-Kriege bezeichnet, gewann der Internet-Explorer aus dem Hause Microsoft konstant Marktanteile hinzu. Hauptursache war die Integration des Internet-Explorers in die neuen Microsoft Betriebssysteme, die somit nach der Installation schon über einen Web-Browser verfügten. Das zuvor notwendige separate Installieren des Web-Browsers, das viele Endbenutzer nach Qualitätskriterien zugunsten von Netscape hatte entscheiden lassen, konnte nun entfallen. Flankiert wurde dies durch unterschiedliche Versuche Microsofts, über selbst festgelegte Standards Internet-Seiten nur noch über den Internet-Explorer verfügbar zu machen.

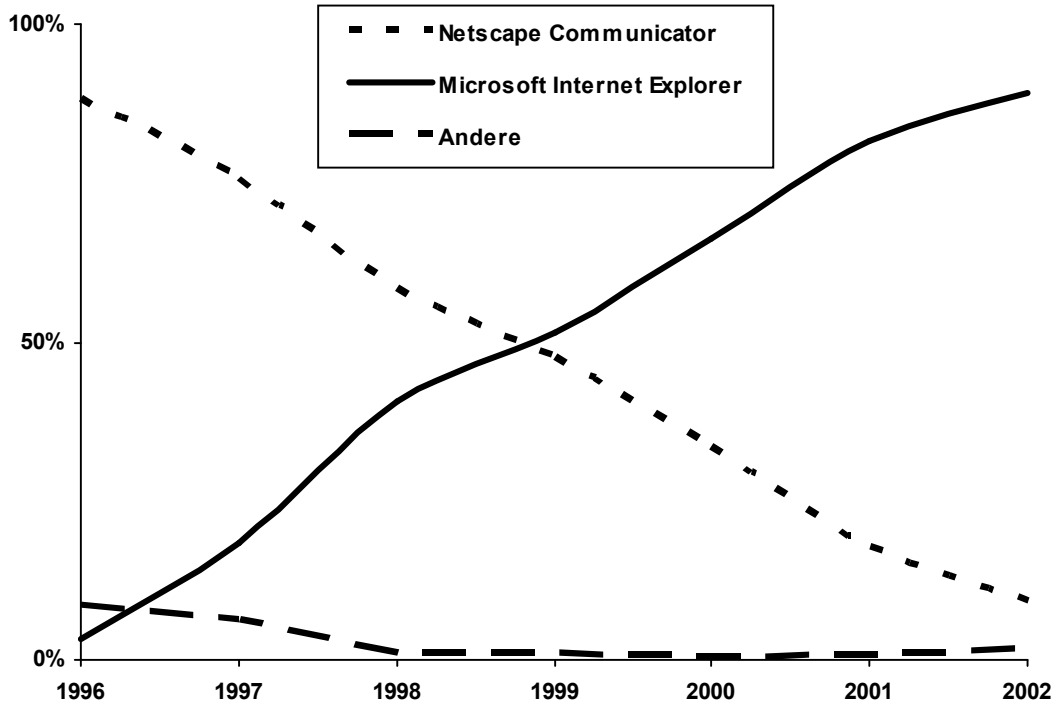

Abbildung 3.1: *Entwicklung der Browser-Nutzung 1996-2002 (Quelle: Browserwatch)*

Die Konsequenz ist, dass der Internet-Explorer heute 80-90% Marktanteil besitzt, während Netscape nur noch als Randerscheinung auftritt. Dennoch ist die Browserlandschaft

nicht so homogen, wie es die Zahlen erscheinen lassen. So teilt sich der hohe Marktanteil des Internet-Explorers auf unterschiedliche Versionen des Browsers auf, wobei die Versionen 4, 5 und 6 das Gros ausmachen. Problematisch ist hierbei, dass in den unterschiedlichen Browser-Versionen zum Teil auch Standards verschieden umgesetzt werden.

In Bezug auf die Entwicklung von Web-Applikationen stellt sich natürlich die Frage nach der Relevanz des Web-Browsers. Wie wir bei der Vorstellung der clientseitigen Techniken noch sehen werden, sind diese allerdings stellenweise in hohem Maße vom verwendeten Browser abhängig. Eine Technik, die im Internet-Explorer 6 läuft, wird eventuell in früheren Versionen nicht oder anders umgesetzt. Auch muss beachtet werden, dass in bestimmten Anwendergruppen ganz andere Browserverteilungen gelten. So ist zum Beispiel für Linux kein Internet-Explorer verfügbar, so dass Web-Applikationen, die auch auf diesen Rechnern benutzt werden sollen, auf jeden Fall auch unter Netscape oder anderen Linux-Browsern lauffähig sein müssen.

3.1.2 Clientseitige Techniken

Browser werden hauptsächlich zur reinen Präsentation der Web-Applikation verwendet. Es besteht aber auch die Möglichkeit, mit unterschiedlichen Techniken Applikationslogik innerhalb des Browsers ausführen zu lassen. Einige von ihnen sollen im Folgenden vorgestellt werden.

HTML

Die Hypertext Markup Language (HTML) ist eine reine Beschreibungssprache. Sie bietet also insbesondere keine Möglichkeit, Programme innerhalb des Browsers zu erstellen. Dennoch lassen sich mit den weiter unten erläuterten serverseitigen Techniken umfangreiche Web-Applikationen entwickeln, bei denen clientseitig lediglich HTML verwendet wird. Zudem ist HTML momentan die einzige Technik, die sämtliche Browser problemlos verarbeiten können, auch wenn es hier zu geringen Unterschieden in der Darstellung kommen kann. Sie ist somit der kleinste gemeinsame Nenner, der innerhalb der Browser einsetzbaren Techniken, auf den man sich bei der Entwicklung von Web-Applikationen verlassen kann.

Das Beschränken auf HTML hat zur Folge, dass sich die recht leistungsstarken Client-Rechner lediglich zur Anzeige von Web-Seiten benutzen lassen. Um das hier vorhandene Potential besser nutzen zu können und um interaktivere Web-Applikationen zu ermöglichen, wurden in den letzten Jahren verschiedene Techniken entwickelt, die im Folgenden erläutert werden sollen.

Applets

Einer der ersten Versuche hierzu waren die von Sun auf Grundlage der Programmiersprache Java entwickelten Applets. Hierbei handelt es sich um nur innerhalb eines Browsers

lauffähige Java-Programme. Standardmäßig besitzen sie auf dem Client lediglich geringe Rechte. So haben sie keine Zugriffsmöglichkeit auf das Dateisystem und können Internet-Verbindungen nur zu dem Rechner aufbauen, von dem sie heruntergeladen wurden. Durch Sicherheits-Zertifikate, die der Benutzer bestätigen muss, können diese Rechte erweitert werden, so dass sich Applets durchaus zu eigenständigen Applikationen ausbauen lassen. Zum Ablauf der Applets ist auf dem Client-Rechner eine installierte Virtual Machine (VM) notwendig, welche die Laufzeitumgebung des Applets darstellt.

Hierin liegt die große Schwäche der Applets, da in unterschiedlichen Browsern durchaus unterschiedliche VM eingesetzt werden. Dies führt dazu, dass Applets heute nur noch selten in Web-Applikationen verwendet werden. Meist sind sie auf bestimmte Funktionalitäten spezialisiert, wie z.B. zur Anzeige vom Browser nicht unterstützter Datei-Formaten oder zur Unterstützung eines komfortablen Datei-Uploads.

ActiveX

ActiveX war Microsofts Antwort auf die von Sun eingeführten Applets. Es handelt sich hierbei um einen Bestandteil aus Microsofts Komponentenmodell (COM), der es ermöglicht, komplexe Kontroll- und Multimediaelemente in Web-Seiten einzubinden. Im Unterschied zu Applets können sie mit unterschiedlichen Programmiersprachen erstellt werden, sind jedoch nicht in jedem Browser verwendbar.

Durch die hohe Verbreitung des Internet-Explorers und der Microsoft-Betriebssysteme sind ActiveX-Komponenten heute im Web deutlich häufiger anzutreffen als Applets. Jedoch stellen sie in der Regel spezialisierte Komponenten dar, wie z.B. den Windows MediaPlayer und dienen nicht zur Umsetzung von kompletten Applikationen.

Die nicht endenden Sicherheitsprobleme haben zudem dazu geführt, dass ActiveX-Komponenten von vielen Benutzer standardmäßig abgeschaltet sind, da dies den einzigen sicheren Schutz vor Missbrauch darstellt.

JavaScript und JScript

JavaScript ist eine von Netscape 1995 zunächst unter dem Namen LiveScript entwickelte Skriptsprache, die eingebettet in HTML-Seiten von Browsern interpretiert und ausgeführt werden kann. Im Zuge des aufkommenden Java-Hypes entschied man sich im Einvernehmen mit Sun in die Umbenennung in JavaScript, was auch heute noch zu Missverständnissen führt. So hat JavaScript zwar einige Konzepte von Java übernommen, ist aber eine eigenständige Sprache mit deutlich reduziertem Funktionsumfang.

Zielsetzung von JavaScript war es, den bis dahin statischen HTML-Seiten innerhalb der Browser mehr Funktionalität zu verleihen. Hierdurch sollten Webseiten zu Programmen werden, die innerhalb des Browsers ablaufen, um auf der einen Seite die Server zu entlasten. Auf der anderen Seite sollten sie zu schnelleren Reaktionszeiten für den Benutzer sorgen.

Durch die schnelle Verbreitung und Akzeptanz aufgeschreckt, führte Microsoft die vergleichbare Skriptsprache JScript und das auf der firmeneigenen Programmiersprache Visual Basic basierende VBScript ein. Während VBScript, da es nur innerhalb des Internet-Explorers läuft, inzwischen kaum noch Verwendung findet, wurde JScript über die Jahre weiterentwickelt.

Zwar richten sich sowohl JavaScript als auch JScript in ihren aktuellen Versionen nach Standards, setzen diese jedoch nur teilweise um. Dies hat zur Folge, dass JavaScript-Anwendungen, die sowohl unter Netscape- als auch Microsoft-Browsern laufen sollen, sehr umständlich zu programmieren sind, da hierbei eine Unmenge von Fallunterscheidungen gemacht werden müssen.

Zudem führen Sicherheitsprobleme auch bei JavaScript dazu, dass der einzig wirksame Schutz des Benutzers im Abschalten dieser Funktionalität im Browser liegt. Somit kann sich der Entwickler einer Web-Applikation beim Einsatz von JavaScript nicht auf dessen clientseitigen Ablauf verlassen, sondern muss die hiermit erstellte Funktionalität auch serverseitig bereitstellen. Dies führt dazu, dass JavaScript in der Regel nur zum Einsatz kommt, wenn es ohne großen Aufwand zur Laufzeit erstellt werden kann und der Nutzen seines Einsatzes in erträglichem Verhältnis zum Aufwand steht.

Wenn JavaScript innerhalb von Web-Applikationen verwendet wird, dann meistens zur clientseitigen Validierung von Eingabedaten. Durch seine Eigenschaft als Skriptsprache kann es hierfür serverseitig zur Laufzeit generiert und eingebettet in die HTML-Seite zum Client gesendet werden. So kann man z.B. vor dem Senden eines Formulars an den Server prüfen, ob benötigte Daten eingegeben wurden und gegebenenfalls eine Fehlermeldung anzeigen. Ohne den Einsatz von JavaScript müssen diese Validierungen alle serverseitig vorgenommen werden, was hier natürlich zu deutlichen höherem Lastaufkommen führt.

3.1.3 Bedeutung für die Entwicklung von Web-Applikationen

Wie schon erwähnt, ist HTML die einzige Technik, die garantiert von nahezu allen Clients interpretiert werden kann. So verzichten die meisten Web-Applikationen auch auf den Einsatz weiterer Techniken, zumal mit den unten vorgestellten serverseitigen Techniken eine Vielzahl von Anforderungen auch durch die Verwendung von HTML erfüllt werden können.

3.2 Web-Server

Nachdem das Internet zunächst als System verteilter Hypertext-Dokumente begonnen hatte, wurden sehr schnell seine Möglichkeiten als Applikations-Plattform erkannt. Zum einen versuchte man, über die oben beschriebenen Techniken clientseitige Applikationen zu erstellen, zum anderen entwickelte man Erweiterungen für Web-Server, mit denen es nun möglich war, neben statischen auch dynamische Inhalte erzeugen zu können.

3.2.1 Übersicht

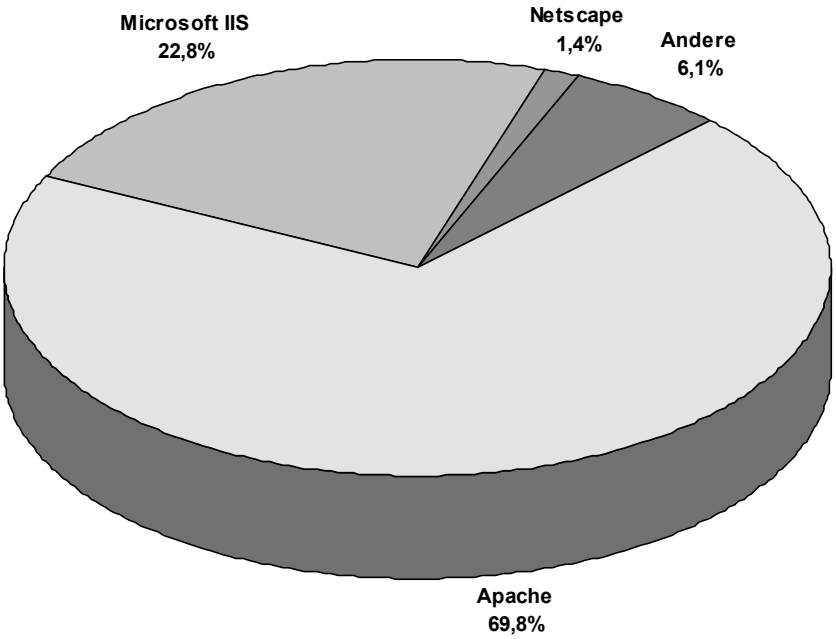

Abbildung 3.2: *Marktanteile Web-Server Mitte 2002 nach Domänen(Quelle: NetCraft)*

Die Web-Server-Landschaft stellt sich sehr übersichtlich dar. Auf der einen Seite steht der Open-Source-Server Apache, auf der anderen Seite die kommerzielle Lösung von Microsoft, der Internet Information Server (IIS). Während der Apache für nahezu jedes Betriebssystem verfügbar ist, ist der IIS lediglich auf Windows-Systemen lauffähig.

Dies ist sicherlich auch der Hauptgrund für den hohen Marktanteil des Apache, neben seiner außerordentlichen Stabilität und Performanz. Der IIS, dessen Marktanteil sich in den letzten Jahren konstant vergrößerte, wird bevorzugt bei der Verwendung von Microsoft Techniken wie ActiveX oder ASP eingesetzt, viele dieser Techniken erzwingen geradezu seinen Einsatz.

Beide Server lassen sich nun durch den Einsatz von serverseitigen Techniken erweitern, die zur Erstellung von Web-Applikationen genutzt werden können. Im Folgenden soll hierüber ein kurzer Überblick gegeben werden.

3.2.2 Serverseitige Techniken

Die Beschreibung der Techniken stützt sich überwiegend auf den Artikel von Pleesel/Wilde [PW01].

CGI

Das Common Gateway Interface (CGI) war eine der ersten Techniken zur Erweiterung der Server-Funktionen. Wie der Name schon erahnen lässt, stellt es keine Sprache dar, sondern definiert eine Schnittstelle, über die der Server auf CGI-Skripte zugreifen kann.

CGI-Skripte können neben HTML auch beliebige andere Inhalte, wie z.B. Grafiken, dynamisch erzeugen. Die Grundidee ist hierbei sehr einfach: Anhand des HTTP-Requests erkennt der Server das auszuführende Skript, übergibt ihm den Request, startet es und sendet den von ihm erzeugten Output an den Client zurück. Die hierbei verwendeten Skripte können in beliebigen Sprachen implementiert werden, vorausgesetzt sie können mit der entsprechenden Schnittstelle kommunizieren.

CGI wird von fast allen Web-Servern unterstützt. Ihm fehlen jedoch einige für Web-Applikationen benötigte Funktionen, wie z.B. Session-Management. Zudem besitzt es eine äusserst schlechte Performanz, da für jede eingehende Frage ein separater Prozess gestartet werden muss, was bei vielen gleichzeitigen Anfragen zu hohen Serverlasten führt.

Um diesen Performanznachteil auszugleichen, wurde mit FastCGI eine Erweiterung entwickelt. FastCGI steigert die Performanz von CGI, indem ausgeführte Prozesse nicht beendet werden, sondern vielmehr auf die nächste Anfrage warten. Somit entfällt bei einer erneuten Anfrage das zeitintensive Erstellen eines Prozesses, was die Performanz deutlich steigert. Jedoch kommt es wieder zu Einbrüchen, wenn ein Prozess mehrfach zeitgleich angefragt wird, weil dann wieder mehrere Prozesse erzeugt werden müssen.

Gemeinsam ist den CGI-Ansätzen zudem das Problem, dass dynamische und statische Inhalte nicht getrennt bearbeitet werden können, sondern innerhalb der Skripte erzeugt werden. Die so oft gewünschte Trennung von Design und Applikationslogik ist somit nicht realisierbar.

PHP

PHP Hypertext Preprocessor (PHP) ist eine Skriptsprache, die sich Elementen aus Perl und C bedient. Mit ihr kann Programm-Code zusammen mit gewöhnlichem HTML in Web-Seiten gemischt verwendet werden. Bei einer Anfrage an eine PHP-Seite wird dann der Programm-Code ausgeführt. Somit kann, wie bei allen Skriptsprachen, die Programmierung an der Stelle erfolgen, an der auch später ihre Ausgabe erfolgen soll.

PHP stellt hierbei eine Vielzahl von Funktionen zur Verfügung, die den Entwickler webspezifische Probleme, wie z.B. die Datenbankanbindung, das Versenden von Emails oder

das dynamische Erstellen von Grafiken, mit geringem Aufwand lösen lässt.

Vorteil von PHP ist die Einfachheit der Sprache, die es dennoch ermöglicht, leistungsfähige Web-Applikationen zu erstellen. Zudem werden häufig benötigte Netzwerkdienste, wie z.B. Datenbanken oder Email, hervorragend unterstützt. Schwächen zeigen sich bei der Integration externer Anwendungen, wie z.B. Unternehmensanwendungen sowie in der Vermischung von Design und Implementierung innerhalb der Skriptseiten.

ASP

Active Server Pages (ASP) ist Microsofts Technik zum serverseitigen Skripting. Zur Implementierung von ASP kann entweder das auch clientseitig verwendete JScript oder Visual Basic Script (VBScript), das auf Microsofts populärer Programmiersprache Visual Basic basiert, verwendet werden. Die eigentliche Stärke von ASP liegt jedoch nicht in den verwendeten Skriptsprachen, sondern in der Möglichkeit über diese Sprachen Softwarekomponenten in die Seiten einzubinden, die dann zur Laufzeit vom Server ausgewertet werden. Microsoft und andere Firmen bieten eine Vielzahl an Komponenten an, die es erleichtern, webspezifische Funktionen, wie z.B. die Generierung von Antwortseiten oder das Ansprechen von Datenbanken, in die Seiten einzubinden.

Die Stärken dieser Technik liegen im Einsatz innerhalb von Microsoft-Umgebungen, da hier sehr leicht weitere Microsoft-Produkte integriert werden können. Größte Schwäche ist die Beschränkung auf den IIS von Microsoft, als einzigen Server der ASP unterstützt. Zudem sind die verwendeten Skriptsprachen sehr im Umfang beschränkt, so dass komplexere Anwendungen nur über Komponenten erstellt werden können.

JSPs und Servlets

Java Server Pages (JSP) sind die Java-basierte Antwort von Sun auf Microsofts ASPs. Ähnlich den anderen Skriptsprachen kann auch hier Programmcode mit HTML vermischt verwendet werden. Als Programmiersprache wird Java verwendet und innerhalb der JSP kann auf große Teile von Java zurückgegriffen werden. JSP stellen somit keine neue Programmiersprache dar, sondern sind vielmehr eine komfortable Möglichkeit, auf Grundlage von Java dynamische Web-Seiten zu erstellen. Ein Alleinstellungsmerkmal von JSP gegenüber anderen Skriptsprachen ist die Möglichkeit, eigene Tags erstellen und in Tag-Bibliotheken verwalten zu können. Hierdurch können komplexe Funktionalitäten gesondert programmiert und über leicht zu verwendende Tags in die Skriptseiten eingebunden werden.

Bei Servlets handelt es sich um Java-Klassen, die als kleine lauffähige Programme in einem entsprechenden Server eingesetzt werden können. Im Unterschied zu JSP werden sie wie gewöhnliche Java-Klassen programmiert. Die hierbei verwendbaren Klassenbibliotheken ermöglichen einen komfortablen, objektorientierten Umgang mit webspezifischen Aufgaben, indem sie z.B. Klassen zur Behandlung von Requests oder zur Verwaltung von Sessions bereitstellen.

Im Unterschied zu CGI können von einem Servlet-Prozess auch mehrere Anfragen parallel bearbeitet werden. Nachteil bei der Verwendung von Servlets liegt in der Tatsache, dass die Ausgaben innerhalb des Servlets generiert werden müssen, was schon bei HTML-Seiten geringeren Umfangs zu umständlichen und schwer wartbaren Code führt.

Daher werden bei Web-Applikationen JSP und Servlets häufig kombiniert eingesetzt. Während die JSP zur Generierung von HTML-Seiten genutzt werden, wird die eigentliche Anwendungslogik in die Servlets ausgelagert. Allein durch diese Trennung wird der Einsatz unterschiedlicher Entwurfsmuster, wie z.B. das später ausführlich vorgestellte Modell-View-Controller, ermöglicht.

Voraussetzung für den Einsatz von JSP und Servlets ist die Unterstützung durch den Web-Server. Entweder ermöglicht er den direkten Einsatz beider Techniken oder er muss dafür nachgerüstet werden.

Vorteile von JSP und Servlets liegen in der durchgehenden Verwendung von Java als gemeinsamer Programmiersprache. Im Zusammenhang mit den weiter unten vorgestellten Application-Servern besteht somit die Möglichkeit, basierend auf Java unternehmensinterne Anwendungen auch über das Internet verfügbar zu machen. Zudem erlauben JSP und Servlets, bei der Erstellung von Web-Applikationen schon innerhalb des Web-Servers objektorientierte Vorgehensweisen anwenden zu können. Als Nachteil ist sicher der hohe Einarbeitungsprozess in Java anzusehen. Im Gegensatz zu z.B. PHP genügt es nicht, sich einzelne Befehle anzuschauen, vielmehr ist ein zumindest grundlegendes Verständnis des hinter Java stehenden Konzepts des objektorientierten Programmierens notwendig. Ein weiterer Kritikpunkt ist die gegenüber PHP deutlich schlechtere Performanz.

3.2.3 Bedeutung für die Entwicklung von Web-Applikationen

Mit Ausnahme von ASP lassen sich die hier beschriebenen Techniken in beiden Servern einsetzen. Stellenweise kann die jeweils zu verwendende Technik durch das Umfeld, z.B. .NET oder J2EE, der Web-Applikation vorgegeben sein. Ist dies nicht der Fall, so können Kriterien wie die Modularisierbarkeit der Applikation, die Einbindung unternehmensinterner Anwendungen oder der Kenntnisstand der Entwickler herangezogen werden. Auf jeden Fall ist der Einsatz der Technik immer situationsbedingt zu treffen, eine generelle Vorgehensweise ist nicht möglich.

3.3 Application-Server

Durch die Vorteile bei der Verwendung von Web-Applikationen stieg in den 90er Jahren das Bedürfnis, auch die in den Firmen vorhandenen Back-Office-Systeme über das Internet verfügbar zu machen. So sollte einerseits den Kunden und Partnern der Unternehmen die Möglichkeit gegeben werden, auf diese Systeme zugreifen zu können, andererseits sollten Angestellte die Möglichkeit haben, auch ausserhalb der Firma über das Internet auf Geschäftsanwendungen zugreifen zu können [Leh02, Seite 3].

Aus diesen Anforderungen heraus entstand mit den Application-Servern eine neue Server-Gattung zur Lösung der anstehenden Probleme. Heute existieren zwei Plattformen, auf denen Lösungen unter Benutzung von Application-Servern erstellt werden können.

3.3.1 .NET

Zwar vermeidet Microsoft die Benutzung des Begriffs Application-Server, jedoch ist allgemein anerkannt, dass die .NET-Strategie in direkter Konkurrenz zu Suns J2EE steht. Während J2EE eine reine Spezifikation ist, umfasst .NET zusätzlich Produkte, die auf dieser Spezifikation beruhen.

.NET spezifiziert unterschiedliche Schichten, von denen folgende für Web-Applikationen von Interesse sind: ASP.NET, die Fortführung der weiter oben beschriebenen ASP, steht für die Erstellung der Präsentationsschicht zur Verfügung. Hierunter liegt die Schicht der Geschäftslogik, die durch .NET- oder COM+-Komponenten gebildet wird. Die Verbindung zu Datenbanken wird über ADO.NET hergestellt.

Sämtliche .NET-Applikationen nutzen dieselbe Laufzeitumgebung, die Common Language Runtime. Über diese ist es möglich, nahezu jede Programmiersprache innerhalb von .NET zu verwenden.

Der zu J2EE vergleichbare Application-Server wird nun durch das Windows-Betriebssystem zur Verfügung gestellt. Für weitere Aufgabengebiete, wie z.B. Transaktionen, existieren zusätzliche Server. Der Markt beschränkt sich hierbei zur Zeit auf Server von Microsoft.

3.3.2 J2EE

Java 2 Enterprise Edition (J2EE) ist eine Spezifikation, welche Standards zur Implementierung, Konfiguration, Verteilung und Einsatz von unternehmensweiten Anwendungen definiert [TSS01, Seite 1]. Es wird hierbei durchgehend Java als Programmiersprache verwendet.

Die Spezifikationen werden hierbei vom Java Community Process (JCP), dem namhafte Firmen wie IBM, BEA und Oracle angehören, erstellt. Ein Produkt welches J2EE-lizenziert werden soll, muss hierfür die Compatibilty Test Suite (CTS) erfolgreich durchlaufen.

Um sicherzustellen, dass die Spezifikationen auch realisierbar sind, existieren für die CTS Referenzimplementierungen, z.B. für Servlets und JSP der Tomcat-Server.
Auf Grundlage der J2EE Spezifikation konkurrieren nun unterschiedliche Firmen, die ihre Produkte nach J2EE bei Sun lizenziert haben. Der Markt ist hierbei nicht so durchsichtig wie z.B. bei Web-Servern, jedoch gelten IBM und BEA im Bereich der Application-Server als Marktführer.

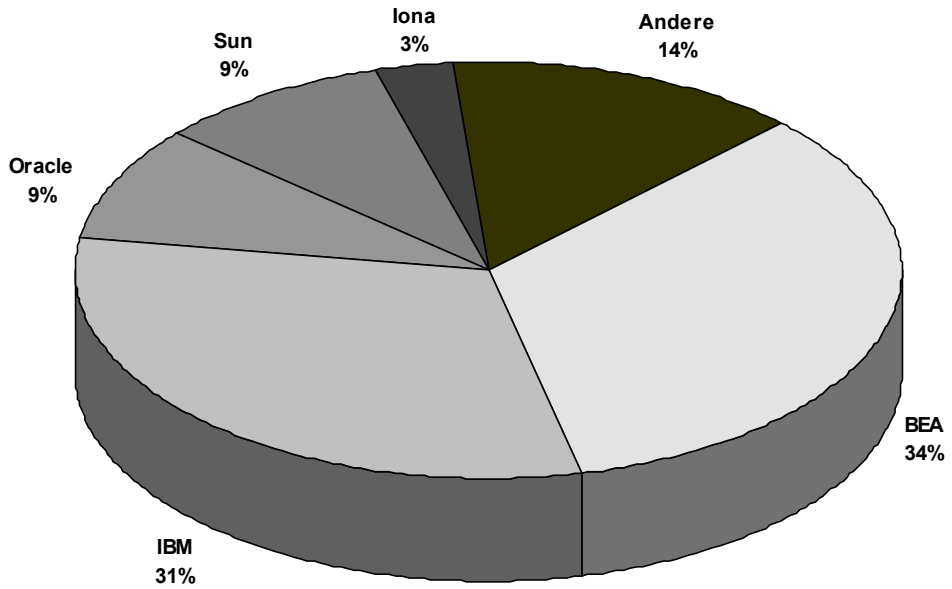

Abbildung 3.3: *Marktanteile J2EE-Application-Server 2001 (Quelle: Gartner Dataquest)*

Marktbeobachter gehen davon aus, dass die Application-Server in den nächsten zwei Jahren verstärkt mit anderen Systemen, wie Betriebssystemen oder Datenbanken, gebündelt werden und nicht mehr allein stehend vermarktet werden [Nat01]. Diese Entwicklung lässt sich sehr gut am Oracle Application Server erkennen, der aus der starken Stellung von Oracle im Datenbankbereich heraus in den letzten beiden Jahren zunehmend Marktanteile gewinnen konnte.

Für die Erstellung von Web-Applikationen sieht J2EE eine Drei-Schichten-Architektur vor, wobei der Schwerpunkt auf der Spezifikation der mittleren Schicht (*Tier*) und der in ihr enthaltenen Container liegt:

1. In der Client-Schicht wird die grafische Benutzeroberfläche zur Verfügung gestellt. Sie kann innerhalb eines Browsers aus Applets und HTML bestehen.

2. In der mittleren Schicht liegt der Web-Container, der, bestehend aus JSP und Servlets, HTML und XML zur Darstellung an die Client-Schicht liefern kann. Im Enterprise Java Beans (EJB) Container wird nun die eigentliche Anwendungslogik

implementiert. Im Unterschied zum Web-Container werden von diesem auch Transaktionen unterstützt.

3. Über die EJB kann nun auch auf die hintere Schicht, die Enterprise Information Systems (EIS) Schicht, zugegriffen werden. In ihr sind Datenbanken und unternehmensinterne Anwendungen enthalten.

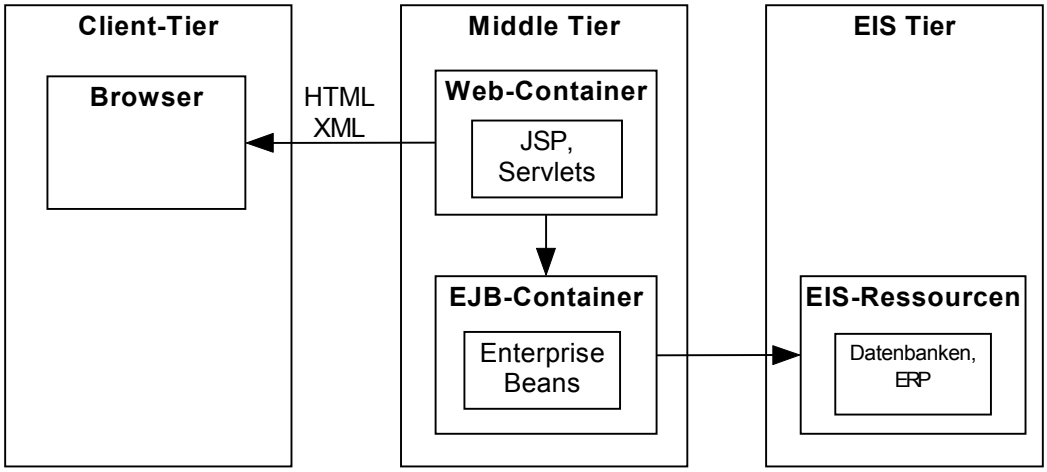

Abbildung 3.4: *Drei-Schicht-Architektur auf Grundlage von J2EE*

Bei weniger umfangreichen Web-Applikationen kann auf die Verwendung des EJB Containers verzichtet werden. Hier greift dann der Web Container direkt auf die Ressourcen der EIS-Schicht zu. Ein solches Vorgehen wird z.B. vom Struts Framework unterstützt, welches sich auf die Verwendung innerhalb des Web-Containers konzentriert. Im Unterschied hierzu ist das Oracle MVC Framework von vorne herein auf die Verwendung beider Container bei der Erstellung von Web-Applikationen angelegt.

3.3.3 Bedeutung für die Entwicklung von Web-Applikationen

Bei der Entwicklung einer Web-Applikation wird man in den meisten Fällen nicht zwischen einer der beiden beschriebenen Plattformen wählen können, da beide sehr tief in die jeweilige Infrastruktur des Unternehmens hineinreichen. Zudem arbeiten beide Plattformen kaum miteinander zusammen, so dass sie die bei Web-Applikationen einsetzbaren Techniken vorgeben.

Speziell bei der Verwendung von J2EE ergeben sich eine Vielzahl von unterschiedlichen Möglichkeiten, wie eine Web-Applikation umgesetzt werden kann. Einige hiervon werden in Kapitel 4 ausführlicher vorgestellt.

3.4 Datenbanken

Nahezu jede zu erstellende Web-Applikation ist zur dauerhaften Speicherung von Daten auf die Verwendung einer Datenbank angewiesen. Bei der Entwicklung der Web-Applikation ist hierbei das Paradigma der verwendeten Datenbank entscheidend. Die Anbindung der konkret verwendeten Datenbanken ist innerhalb eines dieser Paradigmen nahezu identisch, stellenweise können Datenbanken sogar zur Laufzeit ausgetauscht werden. Somit sollen im Folgenden kurz die beiden Paradigmen betrachtet und ihre Auswirkungen auf die Entwicklung von Web-Applikationen erläutert werden. Bei der Beschreibung halte mich an die Erörterung von Starke [Sta02, Seite 127ff].

3.4.1 Relationale Datenbanken

Bei Relationalen Datenbanken (RDBMS) werden Daten in Tabellen abgespeichert. Für die Spalten können unterschiedliche Datentypen festgelegt werden, die eigentlichen Daten werden in den Zeilen abgelegt. Zur Abfrage von Datensätzen steht die standardisierte Structured Query Language (SQL) zur Verfügung. Mit ihr sind auch umfangreiche Abfragen, die über mehrere Tabellen gehen, leicht zu erstellen. Ihre Standardisierung erleichtert zudem die Verwendung einer Web-Applikation mit RDBMS unterschiedlicher Hersteller, wobei jedoch zu beachten ist, dass einige Hersteller SQL eigene Erweiterungen hinzufügen.

Ein Nachteil der relationalen Datenbanken ergibt sich bei ihrem Einsatz mit objektorientierten Programmen. Hierbei kommt es zu einem Strukturbruch, da Objektstrukturen wie Vererbung oder Aggregation nun auf Tabellen abgebildet werden müssen, was weder syntaktisch noch semantisch eindeutig ist. Jedoch haben sich hierfür in den letzten Jahren Strategien entwickelt, die zwar nicht alle Fälle abdecken, sich jedoch in der Mehrzahl als ausreichend erwiesen haben [Sta02, Seite 128].

Dies und die Performanz der relationalen Datenbanken haben zu ihrer starken Verbreitung beigetragen, die sie heute den Markt bestimmen lassen. Bei den kommerziellen RDBMS sind IBM und Oracle als Marktführer zu erwähnen. Demgegenüber steht im Open-Source-Bereich MySQL, welches vor allem in Verbindung mit Linux, Apache und PHP (LAMP), wegen der hohen Performanz zum Einsatz kommt.

3.4.2 Objektorientierte Datenbanken

Die objektorientierten Datenbanken (ODBMS) können sowohl die Attribute und Methoden als auch die Beziehungen von Objekten speichern. Bei ihnen existiert also insbesondere kein Strukturbruch zwischen objektorientierter Programmiersprache und Datenbank. Sie besitzen zur Abfrage der Daten eine Object Query Language (OQL), bei der sich allerdings kein Standard durchsetzen konnte.

ODBMS sind im Markt selten anzutreffen. Ihr Einsatz kann die Zusammenarbeit mit Applikationen, die komplexe Objektstrukturen besitzen, jedoch deutlich vereinfachen und

zu höherer Performanz führen.

3.4.3 Persistenzschicht

Web-Applikationen greifen häufig nicht direkt auf Datenbanken zu, sondern sind von diesen durch eine Persistenzschicht getrennt. Greifen die Objekte der Web-Applikation direkt auf die Datenbank zu, so müssen sie schon bei kleinsten Änderungen, wie zum Beispiel der Umbenennung von Spalten oder Tabellen, angepasst werden. Zudem gestaltet sich das Austauschen der verwendeten Datenbank schwierig, da nun in unterschiedlichen Klassen Quellcode anzupassen ist.

Aus diesem Grund wird häufig eine Persistenzschicht zwischen Web-Applikation und Datenbank eingefügt. In ihr werden alle datenbankrelevanten Details durch eine oder mehrere Klassen gegenüber der Web-Applikation gekapselt. Bei der Entwicklung einer Web-Applikation kommt man somit nicht mehr mit der Datenbank in Berührung, sondern programmiert nun gegen die jeweilige Persistenzschicht, wie z.B. *Enterprise Java Beans* (EJB) oder *Oracles Business Components for Java* (BC4J).

3.4.4 Bedeutung für die Entwicklung von Web-Applikationen

Entscheidend bei der Entwicklung einer Web-Applikation ist also weniger die jeweils verwendete Datenbank, sondern vielmehr deren jeweiliges Paradigma. Hierbei wird man überwiegend auf relationale Datenbanken stoßen, die zudem zum Zeitpunkt der Erstellung der Web-Applikation schon existieren. Der Entwickler muss dann die von der Web-Applikation verwendete Objektstruktur optimal der Datenbankstruktur anpassen, ein Punkt, der sich eventuell stark auf die gesamte Architektur der Applikation auswirken kann.

Bei der Verwendung einer objektorientierten Datenbank oder Persistenzschicht kann diese Problematik weitestgehend verhindert werden.

Kapitel 4

Frameworks

Die im vorigen Kapitel vorgestellten Techniken bieten eine Vielzahl an Möglichkeiten, um Web-Applikationen zu erstellen. Durch die Benutzung von Frameworks wird es dem Entwickler nun ermöglicht, aus diesen Möglichkeiten jene zu wählen, die sich in der Praxis bewährt haben.

Zum Verständnis des Begriffs *Framework* dient der erste Teil, der auch eine Abgrenzung zu anderen Konzepten der objektorientierten Softwareentwicklung vornimmt.

Der zweite Teil beschreibt dann spezielle Applikations-Architekturen, die von Frameworks zur Erstellung von Web-Applikationen verwendet werden.

Im dritten Teil wird schließlich eine Vielzahl von Frameworks vorgestellt, die auf Basis von J2EE zur Erstellung von Web-Applikationen verwendet werden können.

4.1 Allgemeine Betrachtungen

Der Begriff Framework ist schwerer fassbar als andere Begriffe der Software-Entwicklung. Das liegt zum einen darin begründet, dass Frameworks oft nur im Zusammenhang mit den Umgebungen, in denen sie eingesetzt werden, verstanden werden können, zum anderen definieren sie nicht direkt als Einheit erkennbare Regeln und Vorgehensweisen.

4.1.1 Übersicht

Um somit Frameworks in Hinblick auf Web-Applikationen besser verständlich machen zu können, werden zunächst einige Definitionen aufgeführt. Danach werden einige aus diesen Definitionen folgenden Eigenschaften sowie Vor- und Nachteile bei der Verwendung von Frameworks erläutert. Abschließend folgt die Beschreibung der wichtigsten Framework-Arten.

Definitionen

In der Literatur finden sich einige zwar unterschiedliche, jedoch in der Gesamtheit stimmige Definitionen von Frameworks. So ist bei Gamma [GHJV96, Seite 37] zu lesen:

> *„Ein Framework besteht aus einer Menge von zusammenhängenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen.“*

Die bestimmende Eigenschaft eines Frameworks, ist somit die Entwurfsgestaltung. Das Framework bestimmt somit den Kern der Anwendung, lediglich der Code, den das Framework aufrufen soll wird neu entwickelt.

Balzert fügt dieser Definition noch den Aspekt der Anpassbarkeit und Erweiterbarkeit des Frameworks hinzu [Bal00, Seite 843]:

> *„Ein Rahmenwerk (framework) ist ein durch einen Software-Entwickler anpassbares oder erweiterbares System kooperierender Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsbereich implementieren.“*

Ein Framework bestimmt somit die Architektur einer Applikation. Es muss hierfür die Struktur der Klassen und Objekte sowie deren Verantwortlichkeiten festlegen und definieren. Oberstes Ziel ist hierbei immer die Entwurfswiederverwendung, also die Erstellung einer Lösung für Probleme innerhalb eines bestimmten Kontextes [Gie01b, Seite 3].

Vorteile

Ein großer Vorteil liegt natürlich in dem hohen Abstraktionsgrad, mit dem Frameworks die an sie gestellten Anforderungen erfüllen. Hierdurch dienen sie nicht nur der Lösung eines einzigen Problems, sondern können unmittelbar bei einer Klasse von Problemen angewendet werden [Gie01b, Seite 3].

Frameworks entstehen in der Regel aus Erfahrungen, die aus mehreren innerhalb eines ähnlichen Kontextes durchgeführten Projekten stammen. Somit enthalten sie einen sehr großen Erfahrungsschatz, den sie durch ihr Anwenden dem Entwickler unmittelbar verfügbar machen.

Die Entwicklung von Web-Applikationen ist häufig ein iterativer Prozess. Die von der Applikation zu erfüllenden Anforderungen ergeben sich oft erst während der Entwicklung. Ist die Architektur des Frameworks zu früh zu eng angelegt worden, läuft man Gefahr, in eine Sackgasse zu geraten. Frameworks können hier helfen, indem sie die Grundlage zum Aufbau der Applikation sehr breit anlegen und man somit flexibel auf neue Anforderungen reagieren kann.

Zudem besitzen die auf Basis eines Frameworks entwickelten Applikationen eine ähnliche Struktur, was Vorteile für die Verständlichkeit und Wartbarkeit der erstellten Lösungen bietet.

Nachteile

Broy und Siedersleben unterziehen die objektorientierte Programmierung in ihrem Artikel einer kritischen Betrachtung [BS02]. Viele Kritikpunkte die sie aufführen, finden sich auch bei Frameworks wieder. So erwähnen sie unter anderem [BS02, Seite 5]:

> *„Die Objektorientierung bietet keine Operation zur Komposition von mehreren Klassen zu einer zusammengesetzten Klasse. ... Somit unterstützt uns die Objektorientierung nicht bei der Strukturierung von großen Systemen in Komponenten - aber genau das wäre dringend nötig.“*

Natürlich verwenden Frameworks Komponentenbildung zur Erstellung der jeweiligen Architektur. Durch oben beschriebenen Mangel werden hierbei jedoch jeweils unterschiedliche Vorgehensweisen angewendet, wodurch die Komponentenbildung unterschiedlich vorgenommen und eventuell beim Erlernen eines Frameworks neu verstanden werden muss.

Desweitern bemängeln die Autoren die schlechte Beschreibbarkeit von objektorientierten Systemen [BS02, Seite 10]:

> *„Eine der wesentlichen Aufgaben einer Entwurfssprache ist die Beschreibung von Software- und Systemarchitektur. Dies ist wahrscheinlich die größte Schwachstelle heutiger objektorientierter Modellierungssprachen.*
> *Bei UML ist es beispielsweise nicht möglich, ein Komponentenkonzept mit angemessener Granularität festzulegen.“*

Dieser Punkt fällt sehr deutlich bei der Betrachtung der jeweiligen Framework-Dokumentationen auf. In diesen wird UML lediglich zur Beschreibung von Teilaspekten verwendet, eine Beschreibung der gesamten Systemarchitektur ist hiermit nicht möglich. Häufig wird bei der Beschreibung gänzlich auf UML verzichtet und das System wird textuell oder mit

Schaubildern beschrieben.

Nachteile ergeben sich auch aus der hohen Komplexität der Frameworks, die ein Verständnis zusätzlich erschweren. Der Einsatz eines falsch verstandenen Frameworks kann dessen Vorteile sehr leicht zunichte machen.

Nach so viel Kritik sollen abschließend nochmals Broy/Siedersleben zitiert werden, deren Bemerkung sich auch wieder problemlos auf Frameworks beziehen lässt [BS02, Seite 10]:

> *„Trotz der hier vorgetragenen Kritik soll nicht vergessen werden, dass die Objektorientierung viele wertvolle Konzepte und Prinzipien in die Programmentwicklung eingebracht hat. Allerdings sind Verbesserungen erforderlich.“*

4.1.2 Abgrenzung zu anderen Konzepten der objektorientierten Software-Entwicklung

Frameworks sind ein Konzept unter vielen in der objektorientierten Software-Entwicklung. Andere Konzepte sollen hier kurz vorgestellt und gegenüber Frameworks abgegrenzt werden.

Entwurfsmuster

Der zentrale Gedanke der Entwurfsmuster entstammt nicht einem Informatiker, sondern dem Architekten Christopher Alexander [GHJV96, Seite 3]:

> *„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass Sie diese Lösung beliebig oft anwenden können, ohne sie je ein zweites Mal gleich auszuführen.“*

Bei Balzert lesen wir:

> *„Ein Entwurfsmuster (design pattern) gibt eine bewährte generische Lösung für ein häufig wiederkehrendes Entwurfsproblem an, das in bestimmten Situationen auftritt.“*

Gamma u.a. haben aufbauend auf diesem Gedanken in [GHJV96] den Grundstein zur Beschreibung von einfachen und eleganten Lösungen von Entwurfsproblemen gelegt. Viele der von ihnen beschriebenen Entwurfsmuster werden heute in Frameworks verwendet. Gamma nennt auch drei wichtige Aspekte für die Unterscheidung von Frameworks und Mustern:

> Entwurfsmuster sind abstrakter als Frameworks: Sie erläutern aus ihrer Natur heraus Vor- und Nachteile sowie die Konsequenzen eines Entwurfs. Während Frameworks als Code dargestellt werden können und müssen, ist dies bei Mustern nur exemplarisch möglich. Muster müssen bei ihrer Verwendung somit immer neu implementiert werden.

Entwurfsmuster sind kleiner als Frameworks: Während Frameworks häufig ein oder mehrere Entwurfsmuster enthalten, enthalten Entwurfsmuster niemals ein Framework.

Entwurfsmuster sind weniger spezialisiert als Frameworks: Entwurfsmuster sind nie an einen Anwendungsbereich gebunden. Demgegenüber werden Frameworks explizit für bestimmte Anwendungsbereiche entworfen.

Komponenten

Analog der in der industriellen Produktion üblichen Verwendung von Halbfabrikaten, versucht die komponentenbasierte Software-Entwicklung durch Verwendung von vorgefertigten Software-Bausteinen die einfachere, schnellere und preiswertere Herstellung von Anwendungen zu erreichen [Bal00, Seite 856]. Die hierbei verwendeten Komponenten sind definiert als:

> *„Eine Komponente (component) ist ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.“*[Bal00, Seite 856]

Komponenten stehen somit zwischen Anwendungen und Klassen. Während sie im Allgemeinen durch ihren Bausteincharakter kleiner als Anwendungen sind, kapseln sie eine höhere Komplexität und Funktionalität als einzelne Klassen. In Beziehung auf Frameworks lassen sich somit folgende Aussagen treffen:

Frameworks können auf Komponenten zurückgreifen, um sie sinnvoll in einen Anwendungskontext einzuordnen.

Frameworks können benutzt werden um Komponenten zu erstellen.

Der erste Punkt ist bei fast allen weiter unten beschriebenen Frameworks zu finden. Der zweite Punkt ist unter anderem beim Oracle MVC Framework anzutreffen. Hier wird der Entwickler durch die Architektur des Frameworks sowie die von ihm einzuhaltenden Regeln stellenweise dazu gezwungen Komponenten zu erstellen, die dann auch in anderen Applikationen wiederverwendet werden können.

Klassenbibliotheken

Klassenbibliotheken bilden die Voraussetzung, um Wiederverwendung im Kleinen zu realisieren. Sie sind definiert als:

> *„Eine Klassenbibliothek ist eine organisierte Software-Sammlung, aus der der Entwickler nach Bedarf Einheiten verwendet.“*[Bal00, Seite 840]

Im Gegensatz zu Frameworks ist das Ziel von Klassenbibliotheken somit nicht die Entwurfs- sondern die Code-Wiederverwendung. Die Verwendung von Klassenbibliotheken bringt eine Reihe von Vorteilen:

- Durch ihren Einsatz lässt sich Entwicklungszeit sparen.
- Sie werden von Spezialisten mit hohem Fachwissen entwickelt.
- Da sie eingesetzt und getestet wurden, besitzen sie eine hohe Qualität.
- Zudem sind sie oft plattformübergreifend verfügbar.

Demgegenüber sind als Nachteile zu nennen:

- Bei umfangreichen Klassenbibliotheken besteht ein hoher Einarbeitungsaufwand.
- Unterschiedliche Klassenbibliotheken können oft nur durch eine Zwischenschicht gekoppelt werden.
- Beim Einsatz mehrerer Bibliotheken kann es zu Namenskonflikten kommen.

Der letzte Punkt ist bei Java-Bibliotheken selten anzutreffen, da hier die Namensräume standardisiert an eindeutige Internet-Adressen gekoppelt sind.

Klassenbibliotheken werden bei der Entwicklung von Web-Applikationen von allen Frameworks verwendet. Am häufigsten wird hierbei sicherlich auf Suns Servlet-API zurückgegriffen.

4.2 Framework-Architekturen für Web-Applikationen

Wie oben beschrieben, werden durch Frameworks Architekturen festgelegt. Im Folgenden sollen von diesen einige für Web-Applikationen verwendbare vorgestellt werden.

4.2.1 Schichten-Architektur

Beschreibung

Zur Strukturierung von Software-Architekturen hat sich die Schichtenbildung als effizientes Mittel etabliert [Sta02, Seite 98]. Im Folgenden soll nach einem ersten Überblick auf die speziellen Eigenschaften der Schichten-Architektur im Hinblick auf Web-Applikationen und deren Frameworks eingegangen werden. Beim Schichtenmodell stellt

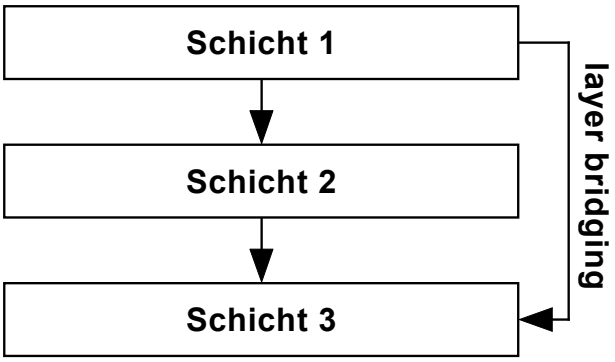

Abbildung 4.1: *Schichten-Architektur*

jede Schicht eine bestimmte Menge von Diensten zur Verfügung. Auf diese Dienste kann jeweils nur von der direkt darüber liegenden Schicht zugegriffen werden. Ihre Komplexität wird dabei von jeder Schicht gekapselt und ist von außen nicht zu erkennen.

Die Schichtenbildung innerhalb einer Software-Architektur bietet diverse Vorteile:

- Die Schichten sind sowohl bei der Entwicklung als auch im Betrieb voneinander unabhängig.
- Unterschiedliche Implementierungen einer Schicht können ausgetauscht werden, sofern sie dieselben Dienste anbieten.
- Die Abhängigkeit von Komponenten wird minimiert. Dies führt zu einer deutlichen Steigerung der Flexibilität und Anpassbarkeit des Systems.
- Keine zu starke Einschränkung des Entwicklers, da er innerhalb der strengen Schichtenhierarchie noch genügend Strukturierungsmöglichkeiten innerhalb der Schichten besitzt.
- Schichtenbildung ist trotz seiner Leistungsfähigkeit ein sehr leicht verständliches Konzept.

Demgegenüber sind einige Nachteile zu nennen:

- Schichtenbildung kann die Performanz eines Systems beeinträchtigen, da z.B. bei einer Anfrage mehrere Schichten zu durchlaufen sind. Man kann diesen Performanz-Verlust durch *layer-bridging*, bei dem Schichten übersprungen werden, vermeiden, jedoch nur auf Kosten zusätzlicher Abhängigkeiten innerhalb des Systems.
- Einigen Änderungen des Systems werden von Schichtenmodellen nur schlecht unterstützt: Wird zum Beispiel eine zugrunde liegende Datenbank geändert und soll sich diese Änderung auch in der Präsentationsschicht wiederfinden, so sind Änderungen an vielen beteiligten Schichten notwendig.
- Allein die Benutzung des Schichtenmodells schützt nicht vor Chaos in den einzelnen Schichten.

Umsetzung bei Web-Applikationen

Ein Standardvorgehen bei der Zerlegung eines Systems in Schichten ist das Zerlegen in die drei folgenden Bestandteile [Sta02, Seite 99]:

Präsentation: Beinhaltet alle für die Anzeige notwendigen Funktionalitäten.

Fachdomäne (Business Architecture): Beinhaltet die fachlichen Aspekte des Systems und ermöglicht es somit dem Entwickler, diese in einer separaten Schicht behandeln zu können. Es ist darauf zu achten, dass keinerlei Aspekte der Fachdomäne in die Präsentationschicht ausgelagert werden, da hierdurch zum einen schwer wartbare Systeme erzeugt werden, zum anderen kann die Wiederverwendbarkeit beider Schichten stark eingeschränkt werden.

Infrastruktur (Technical Architecture): In ihr wird die Komplexität der technischen Infrastruktur vor der Fachdomäne und Präsentation gekapselt. Diese Schicht wird häufig in weitere Schichten oder Komponenten aufgeteilt, um Aspekte wie Persistenz, Sicherheit oder Integration von Fremdsystemen gesondert behandeln zu können. Müssen zudem viele Fremdsysteme und externe Ressourcen in das System integriert werden, empfiehlt sich eine weitere Aufteilung in Integrations- und Ressourcenschicht.

Dieses Vorgehen findet sich auch häufig bei der Verwendung der Architektur innerhalb von Web-Applikationen. Ein Beispiel hierfür ist das Oracle MVC Framework, welches unter Berücksichtigung zusätzlicher Aspekte der MVC-Architektur die Architektur in die oben beschriebenen drei Schichten aufteilt.

4.2.2 Modell-View-Controller

Beschreibung

Die Modell-View-Controller-Architektur (MVC) dient zur Entkoppelung von Darstellung (*View*), Datenhaltung (*Model*) und Ablaufsteuerung (*Controller*) eines Systems [Sta02,

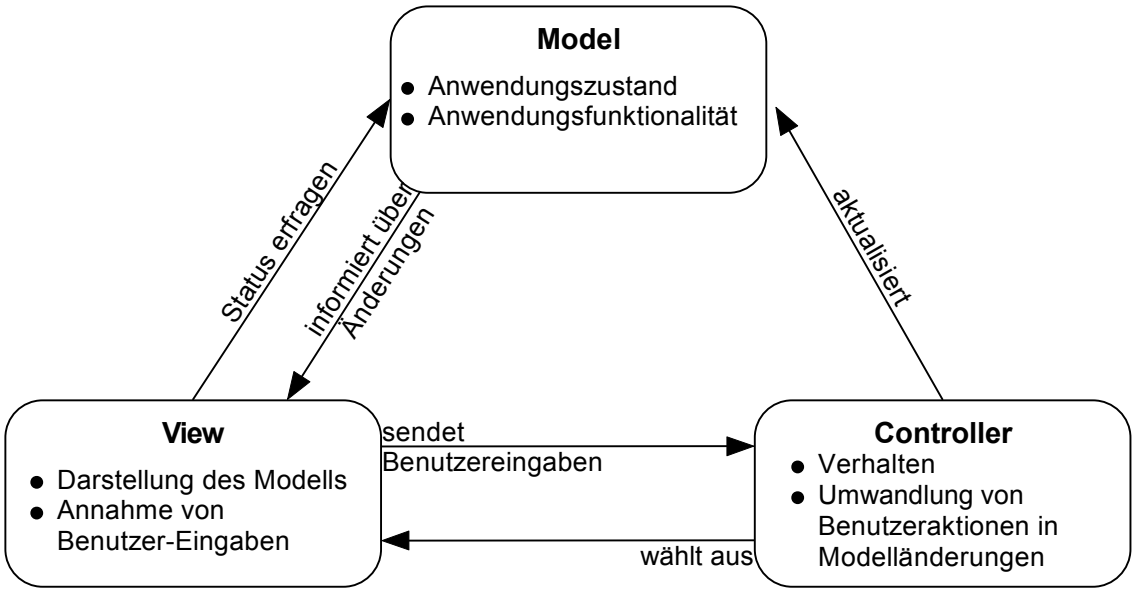

Abbildung 4.2: *Die Modell-View-Controller-Architektur*

Seite 159]. Sie stammt aus der Entwicklung von Smalltalk80-Anwendungen und wird seitdem bei der Entwicklung grafischer Benutzeroberflächen verwendet. Ihre drei Bestandteile haben dabei folgende Funktion:

> **Das Modell** definiert die Funktionalität der Anwendung, und beinhaltet und kapselt deren Zustand. Es beantwortet zudem Fragen zum Zustand und informiert an die Views über Änderungen.
>
> **Die Views** dienen zur Darstellung des Modells, welches sie nach Änderungen befragen können. Zudem nehmen sie Eingaben vom Benutzer entgegen und senden sie zum Controller.
>
> **Der Controller** definiert das Verhalten der Anwendung, also welche Funktionalität des Modells auszuführen ist. Hierzu bildet er Benutzeraktionen auf Modelländerungen ab und wählt den für die Darstellung des Modells notwendigen View aus.

Die lose Koppelung der drei Bestandteile ermöglicht es, diese getrennt zu entwickeln und zu verwenden. So können mehrere Views erstellt werden, die verschiedene Bestandteile des Modells darstellen, ohne dieses hierfür ändern zu müssen. Muss das Modell angepasst werden, weil z.B. Daten anders verarbeitet werden sollen, ist andererseits keine Änderung an den Views notwendig. Und auch innerhalb des Controllers können Änderungen vorgenommen werden, z.B. bei der Auswahl der anzuzeigenden Views, ohne dass diese Auswirkungen auf die anderen Bestandteile haben.

Verwendung bei Web-Applikationen

Durch seine erfolgreiche Verwendung bei der Erstellung von grafischen Benutzeroberflächen wurde schon sehr früh versucht, das MVC-Muster auch bei der Erstellung von

Web-Applikationen zu verwenden. Viele der im J2EE-Umfeld vorhandenen Frameworks basieren auf dem MVC-Muster, wobei jedoch unterschiedliche Schwerpunkte zu erkennen sind.

Aufgrund des verteilten Charakters von Web-Applikationen sind bei der Umsetzung des MVC-Musters allerdings einige Besonderheiten zu erwähnen:

Das Modell unterteilt sich häufig in zwei Bereiche: Einen Back-End-Bereich, der die eigentlichen Geschäftsdaten und -logik enthält sowie einen Front-End-Bereich, der den für die Web-Applikation notwendigen Ausschnitt dieser Geschäftsdaten repräsentiert. Während beide Bereiche bei kleineren Anwendungen durchaus zusammenfallen können, sind sie bei größeren Applikationen in getrennten Komponenten zu entwickeln. Die Verbindung und Aktualisierung zwischen beiden Bereichen wird dann von einem fachlichen Controller übernommen.

Die Views können aufgrund der Beschaffenheit von Web-Browsern und HTTP nicht vom Modell nach Bedarf aktualisiert werden. Vielmehr muss das Modell auf eine erneute Anfrage des Clients warten, um Veränderungen an diesen übermitteln zu können. Zudem ist der Client nicht in der bisher gewohnten Weise kontrollierbar: so kann er z.B. zu einem View mehrere Fenster öffnen, die dann nach kurzer Zeit unterschiedlich aktuelle Zustände enthalten. Es ist für den Ablauf der Web-Applikation eminent wichtig, diesen Punkt zu berücksichtigen und im Controller abzufangen.

Der Controller unterteilt sich bei Web-Applikationen häufig in einen technischen und einen fachlichen Teil. Während die Aufgabe des technischen Controllers die Kontrolle des Workflows innerhalb der Applikation ist, stellt der fachliche Controller die Verbindung zwischen Front- und Back-End-Modell dar.

Beispiele für Frameworks, die die MVC-Architektur in unterschiedlichen Ausprägungen umsetzen, finden sich ab Seite 43.

4.2.3 Pipes und Filter

Beschreibung

Die Pipes-[1] und Filter-Architektur ist besonders gut für Applikationen geeignet, in denen die einzelnen Arbeitsschritte autonom und in einer bestimmten Reihenfolge abgearbeitet werden.

Die Pipes stellen hierbei Datenströme dar, welche an die Ein- und Ausgabe von Filtern angeschlossen werden können. Die Filter können lesend auf die Eingabe- und schreibend auf die Ausgabeströme zugreifen. Innerhalb der Filter können die Datenströme transformiert werden, wobei ein Filter autonom und unabhängig von anderen Filtern arbeitet. Hierdurch sind Filter oft gut wiederverwendbar und es können leicht weitere Filter in das

[1] Rohr, Rohrleitung

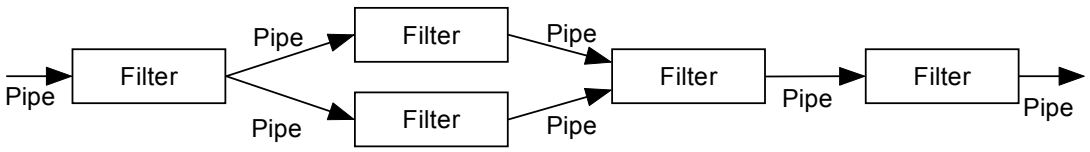

Abbildung 4.3: *Aufbau der Pipes und Filter Architektur*

System integriert werden [Gie01a, Seite 5-6].

Unter Umständen erzwingt die Architektur das häufige Umwandeln von Datenformaten, was zu einer schlechten Performanz führen kann. Sie ist zudem nicht für interaktive Systeme geeignet.

Verwendung bei Web-Applikationen

Das Cocoon Framework (s. Seite 40) beinhaltet eine sehr schöne Umsetzung der Pipeline, eines Spezialfalls der Pipe- und Filter-Architektur. Die Pipeline ist dadurch gekennzeichnet, dass die Filter linear geschaltet sind und es somit nicht zu zyklischen Prozessen kommen kann.

4.2.4 Ereignisgesteuert

Beschreibung

Zentraler Bestandteil dieser Architektur ist der Event-Manager. Objekte können bei diesem Methoden registrieren, die beim Eintreten bestimmter Events ausgeführt werden sollen. Verwendet wird die Architektur für Applikationen, bei denen der Programmablauf nicht von vorne herein feststeht, wie zum Beispiel bei grafischen Benutzeroberflächen oder beim Einsatz von Debuggern. Die einzelnen Komponenten sind hier nur lose innerhalb der Architektur gekoppelt und somit leicht austauschbar. Ein Nachteil ist die Unkenntnis darüber, ob überhaupt ein Objekt auf ein bestimmtes Ereignis antwortet. Zudem ist die Reihenfolge in der Event-Ausführung nicht deterministisch. Da die Semantik zudem vom Zustand des Event-Managers abhängt, sind diese System nur sehr schwer zu testen.

Verwendung bei Web-Applikationen

Ereignisgesteuerte Architekturen sind unter anderem bei Frameworks zu finden, die versuchen, eine zu Swing ähnliche grafische Benutzeroberfläche clientseitig lediglich mit HTML zu erstellen. Hierbei werden die Events serverseitig durch eingehende HTTP-Requests ausgelöst. Beispiel für solche Frameworks finden sich ab Seite 43.

4.2.5 JSP und Servlet-Architekturen

Da sich die in dieser Arbeit behandelten Frameworks auf die JSP und Servlet-Techniken stützen, sollen im Folgenden kurz Architekturen vorgestellt werden, die auf beiden Techniken basieren. Die Beschreibungen sollen zudem das Zusammenspiel der beiden Techniken verdeutlichen und hierdurch das Verständnis der Arbeitsweise und Motivation der Frameworks erleichtern.

Reine JSP-Verwendung

Eine Web-Applikation kann lediglich auf Basis von JSP erstellt werden. Die gesamte Applikationslogik ist dann in den JSP enthalten. Auch Zugriffe auf Ressourcen erfolgen hierbei direkt aus den diesen. Hierdurch ist eine sehr schnelle Implementierung möglich, zudem ist die so erstellte Applikation leicht überschaubar. Allerdings sind hier gravie-

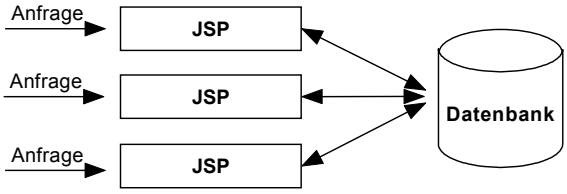

Abbildung 4.4: *Reine JSP-Verwendung*

rende Mängel festzustellen. Durch die hierbei auftretende starke Vermischung von HTML und Java sind die erstellten JSP nur sehr schlecht verständlich und wartbar. Zudem ermöglicht dieser Ansatz keine Skalierung, da die JSP eng miteinander gekoppelt sind. Somit empfiehlt sich der Einsatz dieser Architektur höchstens zur Entwicklung von Prototypen oder Applikationen mit sehr beschränktem Anwendungsbereich [TSS01, Seite 72].

JSP mit Bean

Bei diesem Ansatz, von Sun in früheren JSP-Spezifikationen als *Modell 1* bezeichnet [2], werden Geschäftslogik und Datenzugriffe in Hilfsobjekte, im weiteren als *Bean* bezeichnet, ausgelagert. Hierdurch ist eine Trennung der Entwicklung in einen Design- und einen

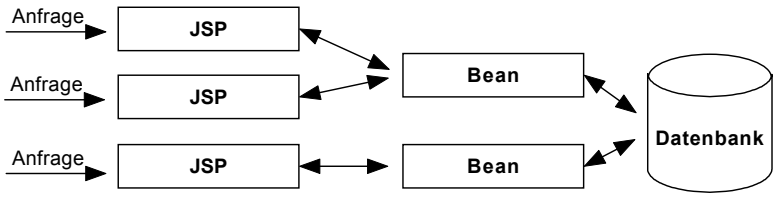

Abbildung 4.5: *JSP mit Bean*

[2] Siehe:JavaServerPages - Specification 0.92: http://www.kirkdorffer.com/jspspecs/jsp092.html

Implementierungsteil möglich. Durch die Entkoppelung der beiden Komponenten wird zudem die Wiederverwendbarkeit deutlich erhöht. So können z.B. die erstellten Beans auch bei mehreren Web-Applikationen, die auf dieselbe Geschäftslogik zugreifen, eingesetzt werden. Jedoch sind auch hier die Beans noch recht stark an die JSPs gekoppelt, was eine gute Skalierbarkeit und Erweiterbarkeit einer erstellten Applikation verhindert. Somit sollte dieser Ansatz nur bei kleineren Applikationen verwendet werden [TSS01, Seite 73].

JSP und Vermittler-Servlet

Hierbei treffen wir zum ersten mal auf beide Techniken innerhalb einer Applikation, weshalb sie von Sun auch konsequenterweise als *Modell 2* bezeichnet wurde. Die HTTP-Requests gehen jetzt nicht mehr an die JSP, sondern werden von einem Vermittler-Servlet entgegengenommen. Dies ermöglicht es, an einem zentralen Punkt in der Applikation

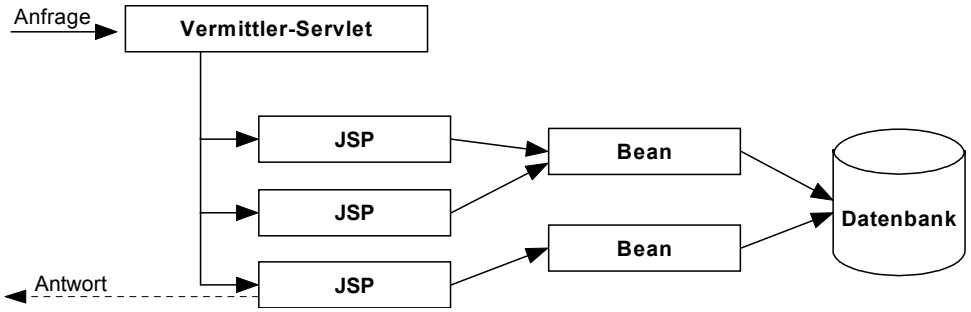

Abbildung 4.6: *JSP und Vermittler-Servlet*

Prüfungen vorzunehmen, z.B. ob der Benutzer der Anwendung eingeloggt ist. Zudem kann das Vermittler-Servlet anhand der eingehenden Anfrage entscheiden, welche JSP angerufen werden soll. Die JSP haben dann wiederum die Möglichkeit, über Beans auf die Geschäftslogik sowie Ressourcen zuzugreifen. Dieser Ansatz empfiehlt sich bei Applikationen die einen zentralen Zugang benötigen, z.B. für Authentifizierung oder Protokollierung. Man kann hier auch schon Ansätze für die Umsetzung der MVC-Architektur erkennen, jedoch sind die Kontrollfunktionen noch in JSPs und Vermittler-Servlet anzutreffen [TSS01, Seite 77f].

Vermittler-Servlet an Worker-Bean

Hierbei erhält man eine Umsetzung der MVC-Architektur. Dazu wird die Kontrollfunktion vollständig aus den JSP entfernt. Das Vermittler-Servlet wählt nun bei einer Anfrage die entsprechenden Worker-Beans aus, in denen weiterhin die Geschäftslogik implementiert ist. Bei umfangreichen Applikationen läuft man Gefahr, dass das Vermittler-Servlet zu umfangreich wird, weshalb sich hier das Einfügen einer Zwischenschicht aus Vermittler-Beans empfiehlt. In diesem Fall wählt das Vermittler-Servlet zunächst die entsprechende Vermittler-Bean aus, die dann wiederum entscheidet, welche Worker-Bean aufgerufen werden soll.

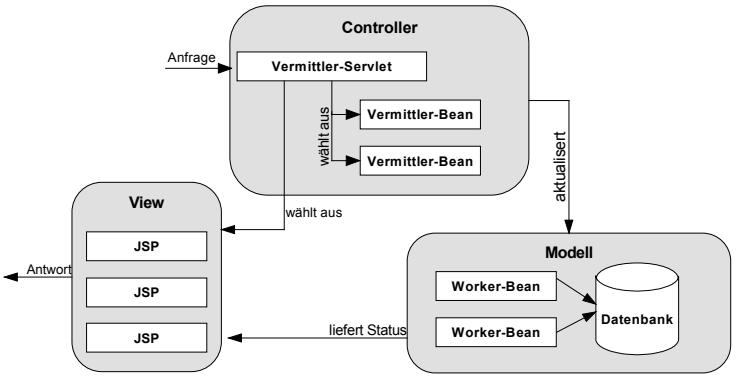

Abbildung 4.7: *Vermittler-Servlet an Worker-Bean*

Festzuhalten ist, dass durch dieses Vorgehen Controller und Modell streng voneinander getrennt werden. Da zudem die JSPs nur noch zur Darstellung des Modells dienen, erhält man so eine konsequente Umsetzung der MVC-Architektur [TSS01, Seite 85].

Das Struts-Framework ist ein Beispiel für die Umsetzung dieser MVC-Architektur.

4.3 J2EE Frameworks

Im Folgenden sollen Frameworks aufgezeigt werden, die Entwickler bei der Realisierung unterschiedlicher Aspekte von Web-Applikationen auf Basis von J2EE helfen. In ihrer Gesamtheit zeigen sie, welche immensen Möglichkeiten in der Verbindung von Java und kreativen Programmierern zur Lösung unterschiedlichster Problemen liegen.

Die hier vorgestellten Frameworks decken ein sehr breites Feld an möglichen Einsatzgebieten ab und sind aufgrund ihrer vielfältigen Eigenschaften nur sehr schwer klassifizierbar. Um dennoch zumindest eine lockere Strukturierung benutzen und einige Gemeinsamkeiten hervorheben zu können, habe ich ihr mögliches Anwendungsgebiet oder spezielle Eigenschaften als Kriterium zur Einteilung gewählt. Viele der hier vorgestellten Frameworks erfüllen durchaus mehrere dieser Kriterien.

Bei der Beschreibung der einzelnen Frameworks beziehe ich mich, wenn nicht anders angegeben, auf die unter der jeweiligen Internet-Adresse verfügbare Online-Dokumentation.

4.3.1 Template-Engines

Template[3]-Engines[4] bezeichnen Frameworks die auf Basis einer eigenen Template-Sprache Änderungen in Dokumenten vornehmen können, um so dynamisch Daten in diese einzubinden. Es können hierüber nicht nur HTML-Seiten sondern auch andere Dokumentarten, wie z.B. PostScript oder auch Quell-Code, erzeugt werden. Tenor bei allen Template-Engines ist die zu starke Durchmischung von Design und Programmierung bei der Verwendung von JSP. Dieser Durchmischung wird durch die Verwendung der Framework eigenen Template-Sprachen entgegengewirkt, die im Vergleich zu JSP zwar deutlich geringeren Umfang besitzen, dafür aber eine größere Kontrolle darüber bieten, was in den HTML-Seiten gemacht werden kann.

Durch ihre vielfältigen Einsatzgebiete sind Template-Engines Bestandteil vieler Projekte und umfangreicherer Frameworks.

WebMacro		
Entwickler: Semiotek	**URL:** www.webmacro.org	**Version:** 1.0.1, 06/2002

Die Philosophie von WebMacro ist die Trennung von Programmierung und Design von Web-Applikationen. Der Designer soll in die Lage versetzt werden, Look und Feel der Applikation gestalten zu können, ohne bei kleinsten Änderungen auf die Hilfe des Programmierers angewiesen zu sein. Im Gegenzug soll es dem Programmierer ermöglicht werden, seine Vorstellungen von guter Software verwirklichen zu können, ohne sich dafür mit HTML beschäftigen zu müssen. Und schliesslich soll sich der Servlet-Entwickler darauf konzentrieren können, wie und womit Anfragen zu beantworten sind und woher

[3] Vorlage, Schablone
[4] Maschine, Motor

die benötigten Daten beschafft werden können, ohne sich mit dem Modell oder HTML beschäftigen zu müssen.

Um diese Trennung umsetzen zu können, bedient sich WebMacro einer eigenen Skript-Sprache. Um Vermischung von Design und Logik schon im Ansatz zu verhindern, ist diese zwar sehr mächtig im Gestalten der Seiten, ermöglicht es jedoch nicht, eigenmächtig Code in die Seiten einzufügen. Zusätzlich ermöglicht ein Class-Analyzer das automatische Verbinden von gewöhnlichen Java-Objekten mit den Templates. Hierdurch ist der Programmierer niemals gezwungen, sich mit der Seiten-Gestaltung auseinanderzusetzen.

WebMacro wird unter anderem bei AltaVista benutzt und ist Bestandteil andere Frameworks, wie z.B. Turbine.

Velocity		
Entwickler: Jakarta	**URL:** jakarta.apache.org	**Version:** 1.3, 06/2002

Velocity[5] besitzt denselben Ursprung wie WebMacro, weshalb sich auch die beiden Template-Sprachen sehr ähnlich sind. Ürsprünglich Bestandteil eines gemeinsamen Projekts, trennten sich die Wege der Entwickler auf Grund von Unstimmigkeiten über die zu verwendende Lizenzart. Das Velocity-Team entschloss sich, die im WebMacro umgesetzten Konzepte beizubehalten, jedoch von Grund auf neu zu programmieren und hierbei unnötigen Ballast über Bord zu werfen. Unterschiede bestehen heute zudem überwiegend in den verwendeten Parsern, die für das Einlesen der Templates benötigt werden.

An einem kleinen Beispiel sollen im Folgenden die Möglichkeiten des Frameworks aufgezeigt werden: Angenommen das Sekretariat der Hochschule gebe alle neu eingehenden Klausurergebnisse direkt in das Informationssystem der Hochschule ein. Zudem können sich die Studenten auf der Internet-Seite der Hochschule einloggen, um so Informationen abrufen zu können. Man entscheidet sich nun dafür, dass den Studenten nach dem Login ihre aktuellen Klausurergebnisse angezeigt werden sollen.

Um dies mit Velocity zu realisieren sind folgende Schritte nötig: Designer und Programmierer setzen sich zusammen und entscheiden, dass in *$student* die Informationen zum momentan eingeloggten Studenten enthalten sind. Zudem sind in *$aktuelleErgebnisse* die Klausuren enthalten, zu denen in den letzten Wochen Ergebnisse eingetragen wurden. Aufgabe der Programmierer ist es jetzt, die notwendigen Objekte zu erstellen. Sie müssen sich dabei jedoch nicht mit Design-Fragen beschäftigen. Auf der anderen Seite können die Designer die vereinbarten Objekte in ihren HTML-Seiten über die Template-Sprache einbinden, ohne sich jemals mit deren Programmierung beschäftigen zu müssen. Ein verwendetes Statement könnte demnach wie folgt aussehen:

```
<HTML><BODY>
Hallo $student.Name.
<table>
```

[5] Schnelligkeit, Geschwindigkeit

```
#foreach( $klausur in $aktuelleErgebnisse )
  #if ( $klausur.hatGeschrieben($student) )
    <tr>
      <td>
        $klausur.Name
      </td>
      <td>
         $klausur.getNote($student)
      <td>
    </tr>
  #end
#end
</table>
</BODY></HTML>
```

Die Verbindung zwischen Template und verwendeten Objekten stellt nun der Class-Analyzer her. Er vermutet z.B. an der Verwendung von *$klausur*, dass ein Objekt *klausur* im Kontext vorhanden ist. Er versucht dann, über den Java-Reflection[6] Mechanismus die Methode *hatGeschrieben(Object o)* aufzurufen.

FreeMarker			
Entwickler: k.A.	**URL:**	freemarker.sourceforge .net	**Version:** 2.03, 06/2002

FreeMarker ist vom Ansatz ähnlich zu den beiden vorangehenden Frameworks. Ein großer Unterschied besteht jedoch in der Verwendung des Class-Analyzers. Objekte auf die innerhalb von Seiten zugegriffen werden soll, müssen bei FreeMarker mindestens eines der Interfaces *freemarker.template.TemplateScalarModel, freemarker.template.TemplateSequenceModel* oder *freemarker.template.TemplateHashModel* implementieren. Beim Parsen des Templates erkennt nun FreeMarker, dass im Kontext ein Objekt *$student* vorhanden ist. Aufgrund der implementierten Schnittstelle muss aber das Objekt nicht über Reflection untersucht werden, sondern es kann auf die in den Schnittstellen deklarierten Methoden zugegriffen werden. Während nun bei WebMacro und Velocity innerhalb der Templates auf alle öffentlichen Methoden des Objekts zugegriffen werden kann, schränkt FreeMarker dies auf die in den Schnittstellen angegebenen ein. Hierdurch soll verhindert werden, dass die Templates in zu großem Maß mit Objekt-Methoden angereichert werden können, was diese aus Sicht der FreeMarker-Entwickler zu stark mit den verwendeten Objekten koppeln würde.

[6]Reflection bietet den Vorteil, den Service von Klassen zu nutzen, wenn es zur Compile-Zeit nicht möglich ist, diese Klassen auf die Einhaltung gewisser Schnittstellen zu überprüfen. ... Reflection hat den inhärenten Nachteil, die Typsicherheit zu unterlaufen, so dass viele Fehler erst zur Laufzeit auftreten und der Compiler keine Chance mehr hat, Fehler frühzeitig abzufangen [Ess01, Seite 474].

4.3.2 XML-Basiert

XML wird durch seinen generellen Ansatz bei fast allen Frameworks in irgendeiner Art und Weise verwendet. Während es jedoch meistens zur Konfiguration oder zum Datenaustausch verwendet wird, benutzen es die beiden nun vorgestellten Frameworks auch an entscheidenden Stellen intern.

Cocoon		
Entwickler: Apache	**URL:** xml.apache.org	**Version:** 2.0.3, 06/2002

Bei der Beschreibung des Frameworks beziehe ich mich überwiegend auf den Artikel von Langham/Ziegeler [LZ02].

Unter Verwendung zahlreicher Komponenten des Apache-Projekts besitzt Cocoon eine offene, komponentenorientierte Architektur. Als Grundlage dient hierbei das Apache Projekt Avalon, welches eine komponentenorientierte Softwareentwicklung ermöglicht. Nach außen kommuniziert das Framework über den auch von Servlets bekannten Request/Response-Mechanismus. Eingehende Requests werden umgewandelt, innerhalb des Frameworks bearbeitet und das Ergebnis als Response zurückgeschickt. Request und Response sind nach außen durch eine Schicht abgeschirmt, welche die vom Framework empfangenen und gesendeten Nachrichten in das erforderliche Format umwandelt. Bei Web-Applikationen handelt es sich hierbei natürlich um die entsprechenden Servlet-Requests und Responses. Es besteht aber auch die Möglichkeit, die Nachrichten mit anderen Umgebungen auszutauschen, wie z.B. der Kommandozeile. Zentraler Bestandteil des Fra-

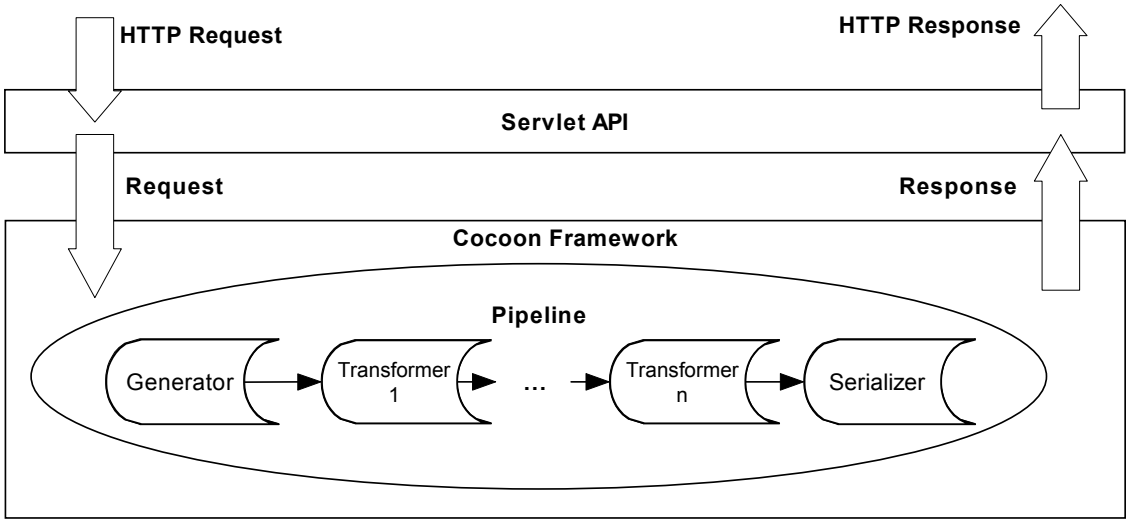

Abbildung 4.8: *Bearbeitung einer Anfrage in Cocoon*

meworks ist die Pipeline. In einer Konfigurationsdatei kann festgelegt werden, welche Arbeitsschritte innerhalb der Pipeline bei Aufruf eines URI ausgeführt werden sollen. Die Pipeline bearbeitet die Anfrage dann in drei Schritten:

1. Zunächst wandelt ein Generator die eingehenden Daten in einen XML-Datenstrom

um. So erzeugt z.B. der *HTML Generator* die XML-Darstellung einer HTML-Seite, der *Directory Generator* die XML-Darstellung eines Verzeichnisses und der *Request Generator* wandelt eine eingehende Anfrage in eine XML-Darstellung um.

2. Im zweiten Schritt überführen nun ein oder mehrere Transformer den oben erzeugten XML-Datenstrom in eine neue Form. Als Ergebnis erhält man also wiederum XML. So wandelt zum Beispiel der *XSLT Transformer* auf Grundlage eines XSLT Stylesheets den eingehenden XML-Datenstrom entsprechend um. Auch hier stehen wieder unterschiedliche Komponenten zur Auswahl, so kann der *SQL Transformer* den eingehenden Datenstrom in eine Datenbankabfrage umwandeln, diese ausführen und das Ergebnis wiederum als XML-Datenstrom darstellen. An den *SQL Transformer* könnte nun der *Filter Transformer* angeschlossen werden, der die ersten 10 Zeilen des Abfrageergebnisses aus dem Datenstrom heraus filtern könnte.

3. Abgeschlossen wird die Verarbeitung in der Pipeline durch die Serializer, deren Aufgabe es ist, den erhaltenden XML-Datenstrom in das gewünschte Ausgabeformat umzuwandeln. Es existieren unter anderem Serializer für HTML, PDF oder auch Microsoft Excel.

Das sehr große Anwendungsgebiet von Cocoon begründet sich nun in der Möglichkeit, die Pipeline über Konfigurationsdateien sehr leicht zusammenbauen zu können. Durch die Vielzahl an vorhanden Komponenten für die einzelnen Arbeitsschritte ergeben sich somit mannigfaltige Kombinationen. Hier wird die Kokon-Idee des Frameworks deutlich, da aus einer auf Cocoon basierenden Anwendung völlig unterschiedliche Ergebnisse aus einer einheitlichen Vorgehensweise erzeugt werden können.

XMLC		
Entwickler: Enhydra	**URL:** xmlc.enhydra.org	**Version:** 2.1, 07/2002

XMLC (Extended Markup Language Compiler) ist ein weiteres Framework, dass sich die Eigenschaften von XML zunutze macht. Wie der Name schon vermuten lässt, hat es den ungewöhnlichen Ansatz, XML- bzw. HTML-Dokumente zu kompilieren.

XMLC unterscheidet bei der Erstellung von Web-Applikationen strikt zwischen den Aufgaben des Designers und des Programmierers. Es wird davon ausgegangen, dass der Designer den Kontakt zum Kunden hält und auf Grundlage dessen Anforderungen das Modell der Applikation anhand eines gewöhnlichen HTML-Editors erstellt. Hierbei verwendet er lediglich HTML 4.0 oder XML, wobei er zur Kennzeichnung von dynamischen Elementen das im HTML 4.0 Standard enthaltene ID-Tag verwendet. Zur Gruppierung der Elemente kann er zudem den ebenfalls im Standard enthaltenen Class-Tag verwenden.Der Programmierer kann nun auf Grundlage der so erstellten Templates mit der Programmierung beginnen. Die erstellten HTML-Templates können hierzu von XMLC kompiliert werden. Anhand der verwendeten ID- und Class-Tags werden die Templates hierbei in Java-Klassen als DOM-Baum integriert. Der Programmierer verwendet nur noch diese Klassen bei der Erstellung der eigentlichen Applikationslogik. Durch diesen Ansatz erreicht XMLC folgendes: Die Templates enthalten nichts anderes als gültiges HTML, ins-

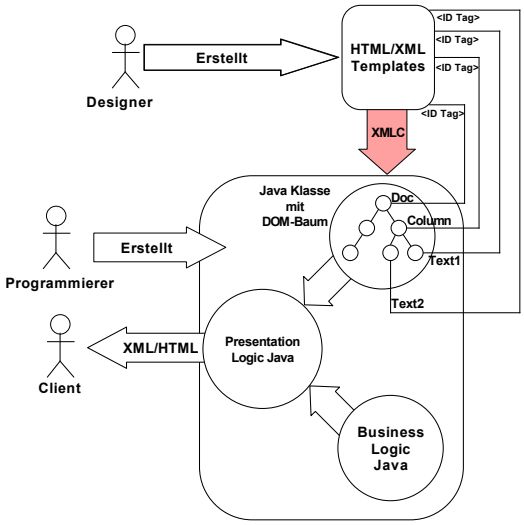

Abbildung 4.9: *Ablauf innerhalb von XMLC*

besondere also keine Skript-Sprachen. Somit befindet sich der Designer zu jedem Zeitpunkt in der für ihn gewohnten Umgebung. Er kann zudem jederzeit das Layout der Seiten verändern, ohne Rücksprache mit dem Programmierer halten zu müssen. Änderungen in den Seiten, die zu Konflikten mit dem Programmteil führen, werden durch das Kompilieren zur Compile-Zeit erkannt. Dies ist sicherlich ein außerordentlicher Vorteil dieses Ansatzes, da diese Fehler normalerweise erst zur Laufzeit der Web-Applikation auftreten und somit äusserst schwer zu entdecken sind. Auf der anderen Seite kann der Programmierer die an ihn gestellten Aufgaben mit Hilfe von Java-Klassen und Objekten erledigen, somit agiert auch er in der für ihn *natürlichen* Umgebung.

In Bezug auf MVC wird der Ansatz von XMLC auch als *push MVC* bezeichnet, da hier das Modell über die Existenz der Views unterrichtet ist und somit in der Lage ist, Informationen in die Views zu *schieben*. Als Nachteil kann hierbei die enge Koppelung des Controllers an die Views angesehen werden[7].

XMLC ist Bestandteil des Barracuda Frameworks, kann jedoch auch selbstständig eingesetzt werden.

[7] http://opensymphony.com/webwork/faq.html

4.3.3 MVC

Das MVC-Muster wurde auf Seite 43 ausführlicher vorgestellt. Die folgenden Frameworks basieren auf diesem Architektur-Ansatz, wobei sie jedoch unterschiedliche Schwerpunkte legen.

WebWork		
Entwickler: OpenSymphony	**URL:** opensymphony.com	**Version:** 1.1, 07/2002

WebWork verwendet im Unterschied zu XMLC einen *pull MVC* Ansatz, bei dem die Views die benötigten Informationen aus dem Modell heraus*ziehen*.WebWork verzichtet hierbei auf die Verwendung der ServletAPI und kann somit auch ausserhalb von Web-Applikationen, z.B. in Swing-Umgebungen eingesetzt werden.
Ziel von WebWork ist es, dem Programmierer die Möglichkeit zu geben, sich innerhalb der vom Framework erstellten MVC-Architektur auf die Erstellung des Controllers konzentrieren zu können. Hierzu wird der Controller durch JavaBeans modelliert, die aus *Set-* und *Get-Methoden* zum Zugreifen auf Attribute sowie einer *execute-Methode* zum Erstellen der Geschäftslogik, bestehen. Zur Erstellung der Views dient eine schlanke Tag-Bibliothek, die lediglich Zugriff und Iterationen für die Daten des Modells enthält.

Struts		
Entwickler: Jakarta	**URL:** jakarta.apache.org	**Version:** 1.02, 02/2002

Struts ist ein Framework, welches basierend auf Suns Modell 2 eine Umsetzung der MVC-Architektur für Web-Applikationen darstellt. Es konzentriert sich hierbei auf die Umsetzung der Views, unter anderem sicherlich wegen der engen Verbindung mit Sun, durch JSPs und deren Zusammenarbeit mit dem Controller, der als Servlet realisiert ist. Das Struts Framework wird ausführlich ab Seite 52 vorgestellt.

Oracle MVC Framework		
Entwickler: Oracle	**URL:** www.oracle.com[8]	**Version:** 2.0RC, 02/2002

Das Oracle MVC Framework ist eine weitere Umsetzung der MVC-Architektur. Es konzentriert sich jedoch hierbei auf die Erstellung des Controllers und ermöglicht es, durch dessen Einbindung in eine schichtorientierte Umsetzung der MVC-Architektur, diesen mit unterschiedlichen Techniken für die Views und das Modell kombinieren zu können. Es legt außerdem großen Wert auf die Wiederverwendbarkeit der erstellten Controller-Komponenten. Das Oracle MVC Framework ist ausführlich ab Seite 83 beschrieben.

4.3.4 Swing-orientiert

Die Swing-orientierten Frameworks versuchen, auf jeweils unterschiedliche Weise die bei der Erstellung grafischer Benutzeroberflächen verwendeten Swing-Klassen für die Erstel-

[8] Es existiert keine eigene Homepage für das Framework. Die Informationen sind etwas verstreut bei Oracle zu finden. Als Startpunkt kann *http://otn.oracle.com/products/ias/ias_utilities.html* dienen.

lung von Web-Applikationen nachzubilden.

Echo		
Entwickler: NextApp	**URL:** www.nextapp.com	**Version:** 0.9.6, 06/2002

Auffallend bei der Verwendung des Echo-Frameworks ist, dass beim Erstellen von Web-Applikationen keine Kenntnisse von HTML und JavaScript nötig sind. Das Framework bietet zu der aus den Swing- und AWT-Bibliotheken bekannten Event-Technik Pendants an, die innerhalb von HTML-Seiten benutzt werden können. Echo unterteilt hierfür die Aufgaben in einen Core- und einen Server-Teil. Der Core-Teil enthält alle Objekte, die zur Gestaltung der Seiten genutzt werden können, wie z.B. Label, Textfelder, Buttons und Fenster. Der hiervon unabhängige Server-Teil besitzt die Aufgabe, die verwendeten Objekte in HTML und JavaScript zu übersetzen. Kommen nun Requests vom Client ein, werden diese vom Server in Events umgewandelt und die entsprechenden Objekte benachrichtigt. Man erhält so eine ähnliche Entwicklungsumgebung wie bei der Gestaltung von Benutzeroberflächen mit Swing oder AWT. Hierzu vergleichbar besteht die Möglichkeit, das Layout der Elemente festzulegen sowie eigene Steuerelemente zu programmieren und im Framework zu benutzen.

WingS		
Entwickler: mercatis	**URL:** wings.mercatis.de	**Version:** 1.0beta, 05/2002

WingS geht einen ähnlichen Weg wie Echo, jedoch bildet es nicht nur das Event-Modell von Swing nach, sondern enthält auch für die meisten Swing-Komponenten ein Pendant mit ähnlicher API und Verhalten. Ähnlich wie in Swing können die Komponenten zudem in Hierarchien angeordnet werden. Durch die starke Anlehnung an Swing können einfache Swing-Anwendungen sehr leicht nach WingS portiert werden. Umgesetzt wird das Konzept in drei Schichten:

1. Die oberste Schicht besteht aus den Swing-ähnlichen Komponenten.

2. In der mittleren Schicht liegt die WingS Session, die konzeptionell mit *java.lang. System* vergleichbar ist. Sie vereint den Dispatcher, der Events an die obere Schicht weiterleitet sowie den Externalizer, der Ressourcen über HTTP verfügbar macht. Pro HTTP-Session existiert immer genau eine WingS-Session.

3. Die unterste Schicht wird schließlich durch die Servlet API dargestellt. Die HTTP-Request sind hierbei mit dem Dispatcher verbunden, während die HTTP-Responses dem Externalizer zugeordnet sind.

WingS besitzt zudem die Fähigkeit, das Erscheinungsbild der Applikation durch die Verwendung von sogenannten LookAndFeel-Klassen anpassen zu können. Diese werden den einzelnen Komponenten zugeordnet und können zur Laufzeit entscheiden, wie diese darzustellen sind. Hierdurch kann das Erscheinungsbild dem Browser des Clients zur Laufzeit angepasst werden. Zudem besteht hiermit die Möglichkeit, neben HTML auch z.B. WML zu unterstützen.

Barracuda		
Entwickler: Enhydra	**URL:** barracuda.enhydra.org	**Version:** 1.1, 06/2002

Im Vergleich zu den anderen hier beschriebenen Frameworks besitzt Barracuda einen deutlich weiteren Fokus. Ein zentraler Bestandteil ist jedoch die Orientierung an Swing, welche innerhalb des Frameworks mit Hilfe von XMLC umgesetzt wird. Die Komplexität von XMLC wird hierbei durch zusätzliche Komponenten gekapselt. Zudem besitzt Barracuda ein eigenes Modell zur Handhabung von HTML-Formularen, mit denen diese Java-Klassen zugeordnet werden können. Weitere Schwerpunkte sind Internationalisierung und Komponentenbildung.

4.3.5 Portale

Aufgabe von Portalen ist die Integration von strukturierten und unstrukturierten Daten aus unterschiedlichen Quellen in eine gemeinsame Web-Oberfläche.

JetSpeed		
Entwickler: Jakarta	**URL:** jakarta.apache.org	**Version:** 1.4b1, 07/2002

Ziel von Jetspeed ist es, den Entwickler von Portalen beim Zusammenstellen und Bereitstellen unterschiedlichster Informationen innerhalb des Portals zu unterstützen. Hauptaugenmerk liegt somit auf der Unterstützung bei der Gestaltung der Web-Oberfläche sowie der Anbindung unterschiedlichster Informationsquellen.
Wichtigster Bestandteil hierfür sind die von Jetspeed verwendeten Portlets. Ein Portlet ist definiert als eine Web-Komponente, die dynamische Inhalte erzeugen kann. Sie sind als Java-Klassen realisiert und können zur Laufzeit in einen Web-Server geladen werden. Im Unterschied zu Servlets interagieren sie nicht direkt mit dem Client, sondern über die Portal-Engine, die hierfür HTTP-Anfragen in Portlet-Anfragen umwandelt. Mehrere Portlets können zusammen innerhalb einer Seite angezeigt werden. Hierbei besitzt jedes Portlet eine den Desktop-Anwendungen vergleichbare *Title Bar*. Zudem besitzt jedes Portlet fünf unterschiedliche Modi, in denen es sich befinden kann:

1. View Modus: Der Standard-Modus, in dem das Portlet Informationen anzeigt.
2. Edit-Mode: Hiermit kann das Erscheinungsbild des Portlets angepasst werden.
3. Maximize: Das Portlet erscheint nun allein im Vollbild.
4. Minimize: Es wird lediglich die *Title Bar* angezeigt.
5. Closed: Das Portlet wird auf der Seite nicht mehr angezeigt.

Jetspeed unterstützt den Entwickler nun zusätzlich, indem es einige Standardportlets zur Verfügung stellt, mit denen Daten schnell in Portlets integriert werden können. So können über das *HTML Portlet* sehr leicht HTML-Seiten oder Fragmente in Portlets eingebunden werden. Natürlich besteht auch die Möglichkeit, eigene Portlets zu erstellen und in das Portal einzubinden.

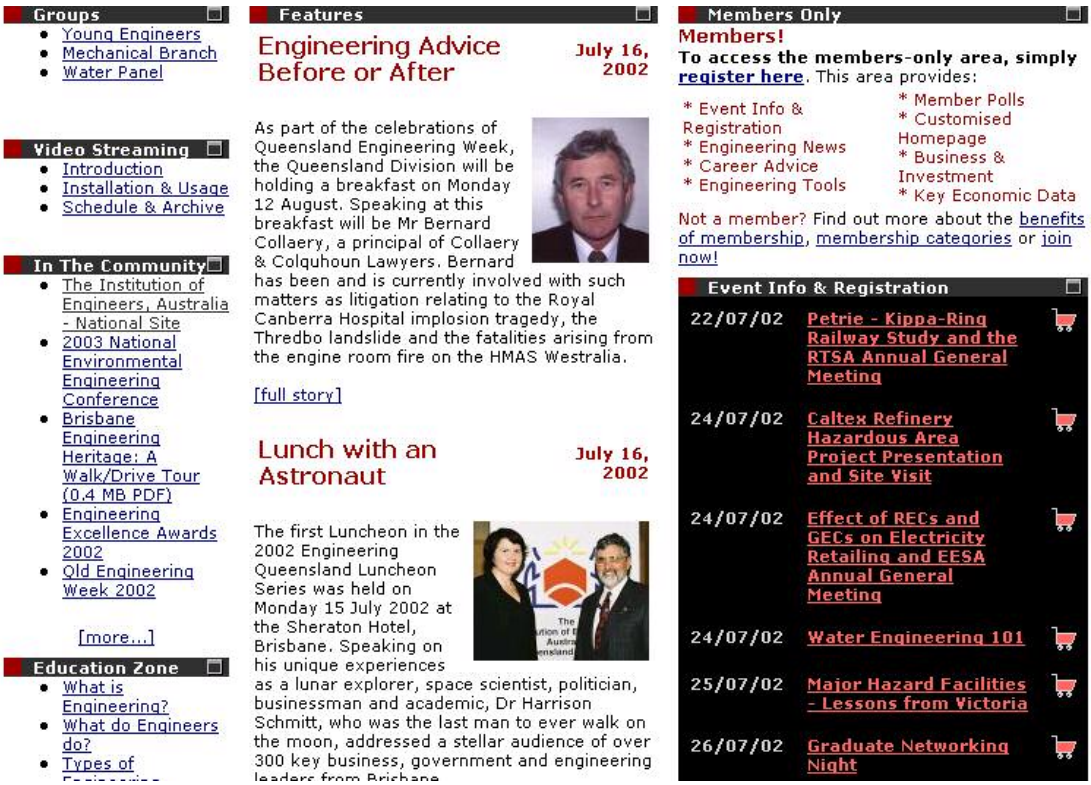

Abbildung 4.10: *Typisches Aussehen eines mit Jetspeed entwickelten Portals (qld.ieaust.org.au)*

Zudem bietet das Framework Unterstützung bei der Benutzerverwaltung, der Personalisierung, sowie dem Caching und Logging. Leider besitzt es kein eigenes Content Management System (CMS), noch eine Suchfunktion [UN02].

InfoZone		
Entwickler: infozone-group	**URL:** www.infozone-group.org	**Version:** k.A.

Infozone ist ein weiteres Framework zur Portalerstellung. Als Kern besitzt es im Unterschied zu Jetspeed mit Prowler[9] ein eigenes Content Management System (CMS)[10]. Prowlers Aufgabe besteht im managen aller verwendeten Ressourcen in einer transaktionsfähigen Umgebung. Als Ressourcen können hierbei unter anderem Mail-Server, Datenbanken oder EJBs eingebunden werden. Auf diesen Ressourcen agieren Adapter, welche die äusserst unterschiedlichen Daten aus den Resourcen nach XML konvertieren. Dies ermöglicht den lediglich auf die Adapter zugreifenden Applikationen, alle benötigten Daten per XML zu laden, zu verändern und zu speichern. Hierbei fasst die Prowler-API die unterschiedlichen XML-Daten zusammen, so dass von der Applikation lediglich auf ein einzelnes XML-Dokument zugegriffen werden muss.

[9] Herumtreiber

[10] Man könnte Prowler auch als transaktionales XML-Filesystem bezeichnen [Brä01, Seite 112].

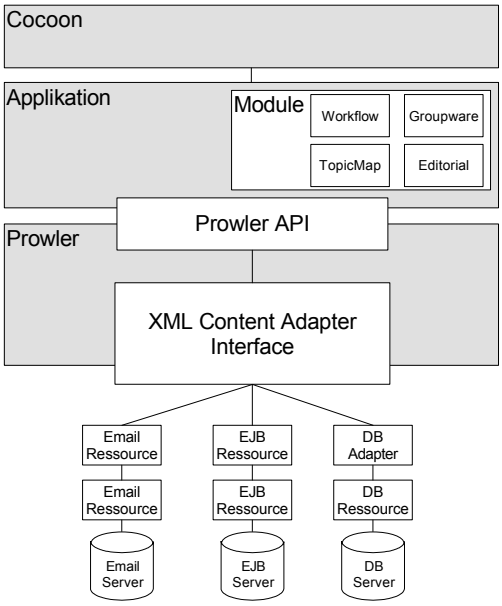

Abbildung 4.11: *Architektur des Infozone Frameworks*

Die gesamte Konfiguration der innerhalb von Prowler verwendeten Ressourcen und Adapter kann hierbei über eine XML-Datei vorgenommen werden, für die eine grafische Benutzeroberfläche zur Verfügung steht. Die auf Prowler ablaufende Applikation kann nun durch unterschiedliche von Infozone bereitgestellte Module erstellt werden. Für die Erstellung der Benutzeroberfläche wird das oben vorgestellte Cocoon verwendet.

4.3.6 Unterstützende Frameworks

Die hier vorgestellten Frameworks befassen sich jeweils mit einem sehr speziellen Aufgabengebiet innerhalb von Web-Applikationen. Während sie somit eigenständig keine große Verwendung finden, sind sie oft Bestandteil größerer Frameworks.

DbForms		
Entwickler: k.A.	**URL:** dbforms.org	**Version:** 1.1.2, 05/2002

DbForms ermöglicht, wie der Name schon erahnen lässt, das schnelle Entwickeln von Web-Applikationen, die zum Zugriff auf Datenbanken dienen. Es deckt damit einen sehr kleinen Bereich ab.

Castor		
Entwickler: Exolab	**URL:** castor.exolab.org	**Version:** 0.9.3, 12/2001

Castor ist ein Data-Binding-Framework. Mit ihm ist z.B. ein einfaches Zuordnen von Java-Objekt-Modellen und XML möglich. Castor kann hierdurch zum einen Java-Objekte in XML ausgeben, zum anderen aber auch Quellcode auf Grundlage von XML-Dateien erstellen.

4.3.7 Generelle Ansätze

Die beiden hier vorgestellten Frameworks vereinen viele häufig benötigte Funktionen bei der Erstellung von Web-Applikationen unter einem Dach. Durch diesen generellen Ansatz sind sie sehr flexibel einsetzbar und können als Basis von unterschiedlichen Arten von Web-Applikationen dienen.

Turbine		
Entwickler: Jakarta	**URL:** jakarta.apache.org	**Version:** 2.2-b2, 07/2002

Innerhalb von Turbine sind momentan ca. 30 Dienste verfügbar, die innerhalb von Web-Applikationen verwendet werden können. Jeder Dienst ist als Singleton realisiert, so dass jeweils nur eine Instanz innerhalb des Systems besteht. Hierdurch ist gewährleistet, dass von den Diensten verwendete Ressourcen, wie z.B. Datenbankverbindungen, zentral verwaltet und gemanagt werden können. Es sollen exemplarisch fünf Dienste aufgezeigt werden, um das weite durch sie abgedeckte Spektrum aufzuzeigen:

- Localization Service: Bietet einen zentralen Zugriff auf alle für die Lokalisierung der Applikation notwendigen Ressourcen.
- Logging Service: Ermöglicht das umfangreiche Loggen innerhalb der Web-Applikation. Es werden die Logging-Mechanismen Log4J, Avalon's LogKit sowie das Logging der Servlet API und ein einfaches Datei Logging unterstützt.
- Scheduler Service: Hierüber können Hintergrundprozesse erstellt und zu bestimmten Zeiten ausgeführt werden, ohne durch einen HTTP-Request angestoßen werden zu müssen. Informationen über die gestarteten Prozesse werden in einer Datenbank gespeichert und können somit auch bei einem Neustart von Turbine automatisch geladen werden.
- Unique ID Service: Hiermit können eindeutige IDs in unterschiedlichen Formaten erzeugt werden.
- Upload Service: Dieser Service ist in der Lage, Uploads über HTTP-Post-Anfragen entgegenzunehmen.

Zusätzlich zu den Diensten bietet Turbine unter Benutzung zahlreicher anderer Frameworks einen Rahmen für die Erstellung von umfangreichen Web-Applikationen. Als Architektur dient hierfür das von den Turbine-Entwicklern kreierte Modell 2+1, eine Erweiterung des Modell 2 von Sun[11]. Hiermit können zum einen in der Präsentationsschicht

[11] Jon Stevens, einer der Entwickler von Turbine, erklärt die Begriffsbildung folgendermaßen: *The +1 was a joke on my part to see what people would do with it. I figured that people were believing this Model 2 hype with JSP and such and I wanted to improve on it. I love playing with marketing games. I read that Model 2 paper and then figured that Turbine was better than what they declared as Model 2. Calling it Model 3 would have been like saying that Turbine was just a third generation of Model 2. So, instead I figured that making it +1 just made it appear to be an improved Model 2 model since that is what Turbine really is. :-)*

andere Techniken als JSP eingesetzt werden, zum anderen sind Methoden wie Event-Handling realisierbar.
Zu den in Turbine verwendbaren Frameworks, die in dieser Arbeit vorgestellt wurden, gehören unter anderem Freemaker, WebMacro sowie Velocity.
Jetspeed ist ein Beispiel für ein auf Basis von Turbine erstelltes Framework.

Expresso		
Entwickler: JCorporate	**URL:** www.jcorporate.com	**Version:** 4.02, 12/2001

Auch Expresso unterstützt den Entwickler mit einer Vielzahl von Komponenten, die je nach Bedarf einzeln zur Erstellung einer Web-Applikation herangezogen werden können:

- Sicherheit: Umfasst sowohl die Authentifizierung der Benutzer als auch die Sicherheit der Applikation vor unberechtigtem Zugriff.
- Connection Pooling: Beim Zugriff auf Datenbanken stellt Expresso eine eigene Komponente zum Verwalten der Verbindungen zur Verfügung.
- Logging: Durch die Integration von Apaches Log4j ist das komfortable Loggen der Applikation möglich. So kann über den Web-Server jederzeit der Zustand des Systems geprüft werden.
- MVC: Expresso besitzt eine eigene Umsetzung der MVC-Architektur, es kann seit Version 4.0 aber auch alternativ Struts verwendet werden.
- Health Check: Hiermit kann der Zustand der Applikation zur Laufzeit überprüft werden. Geprüft werden kann zum Beispiel, ob bestimmte Web-Services laufen oder ob das Email-System noch intakt ist. Bei Störungen des Systems können automatisch Benachrichtigungen an Administratoren versendet werden.

4.3.8 Weitere Frameworks

Im Folgenden führe ich der Vollständigkeit noch weitere Frameworks auf, die entweder zu oben beschriebenen sehr ähnlich sind oder scheinbar nicht mehr weiterentwickelt werden.

Bishop	**Kategorie:** MVC	
Entwickler: Johan Redestig	**URL:** bishop.sourceforge.net	**Version:** 2, 03/2001

H-MVC Framework	**Kategorie:** MVC	
Entwickler: Crionics	**URL:** www.crionics.com	**Version:** 1.0, 09/2001

Cameleon	**Kategorie:** Swing-orientiert	
Entwickler: Müller und Stein	**URL:** www.must.de	**Version:** 1.1, 06/2002

HyperQbs	**Kategorie:** Genereller Ansatz	
Entwickler: HyperQbs	**URL:** www.hyperqbs.org	**Version:** 2.1, 10/2001

Japple	**Kategorie:** Genereller Ansatz	
Entwickler: Saucon	**URL:** japple.org	**Version:** 2.0.5.2, 05/2002

Jade	**Kategorie:** Genereller Ansatz	
Entwickler: salmonLLC	**URL:** www.salmonllc.com	**Version:** 1.01, 06/2002

Mapper	**Kategorie:** MVC	
Entwickler: k.A.	**URL:** mapper.sourceforge.ne	**Version:** 1.9.4, 07/2002

Maverick	**Kategorie:** MVC	
Entwickler: k.A.	**URL:** mav.sourceforge.net	**Version:** 2.1.1, 06/2002

Swinglets	**Kategorie:** Swing-orientiert	
Entwickler: Javelinsoft	**URL:** www.swinglets.com	**Version:** 1.2, 06/2002

Tapestry	**Kategorie:** GUI-orientiert	
Entwickler: Howard L. Ship	**URL:** tapestry.sourceforge.net	**Version:** 2.2, 06/2002

TeaServlet	**Kategorie:** Template-Engine	
Entwickler: Walt Disney IG	**URL:** opensource.go.com	**Version:** 1.4.3, 08/2001

WebCream	**Kategorie:** Swing-orientiert	
Entwickler: CreamTec	**URL:** creamtec.com	**Version:** 4.2.0, 06/2002

Kapitel 5

Das Struts-Framework

In diesem Kapitel soll mit Struts ein Framework zur Erstellung von Web-Applikationen detaillierter vorgestellt werden.

Zunächst wird in der Übersicht das Umfeld sowie die Ziele und die Verbreitung von Struts beschrieben.

Danach werden die einzelnen Bestandteile genauer erläutert und der Ablauf innerhalb des Frameworks erklärt.

Hieran anschließend wird erklärt, wie mit Struts eine Web-Applikation erstellt wird.

Abgeschlossen wird das Kapitel mit einigen kritischen Betrachtungen.

5.1 Übersicht

In Kapitel 4 wurde gezeigt, dass die MVC-Architektur zu den weit verbreiteten Architekturen bei der Erstellung von Web-Applikationen gehört. In diesem Kapitel soll nun mit Struts ein Framework untersucht werden, welches auf dieser Architektur basiert.

5.1.1 Umfeld

Struts ist ein Framework des Jakarta-Projekts [1], welches von der Apache Group, IBM und Sun 1999 mit der Zielsetzung gegründet wurde, eine Referenzimplementierung für Servlets und JSP innerhalb einer Server-Umgebung zu erstellen [HPRT02, Seite 16]. In der Zwischenzeit ist eine Vielzahl von Teil-Projekten aus dem Jakarta-Projekt hervorgegangen, von denen einige in Kapitel 4 beschrieben wurden.

Als Sun die Servlet-Technologie zur Erweiterung von Servern auf den Markt brachte, wurde diese von den Entwicklern zwar gut angenommen, es zeigten sich bei ihrer Verwendung zur Erstellung von Web-Applikationen allerdings auch einige Mängel. Am störendsten war hier sicherlich die Notwendigkeit, HTML-Seiten innerhalb der Servlets, also in den Klassen selbst, implementieren zu müssen. So war es nicht möglich, die benötigten Seiten von Web-Designern erstellen zu lassen.

Um diese Umstand zu beheben, brachte Sun mit JSP eine Technik auf den Markt, die es nun ermöglichte, Java-Code und HTML in einer Skriptseite zu mischen. So konnten die Designer am Erstellen der Web-Applikation mitwirken, die Programmierer hatten die Möglichkeit, dynamische Inhalte in die Seiten einzubinden. Folge hiervon war jedoch, dass durch die Vermischung von Design und Programm schwer wartbare Web-Applikationen entstanden, wodurch zudem die Wiederverwendbarkeit stark eingeschränkt wurde.

Sun reagierte erneut und stellte unterschiedliche Modelle zur Erstellung von Web-Applikationen auf Grundlage beider Techniken vor. Allerdings waren auch diese Modelle nicht sehr ausgereift und brachten nicht den erhofften Nutzen. Die damals verwendeten Bezeichnungen Modell 1 und Modell 2 werden von Sun zwar heute nicht mehr benutzt, dennoch tauchen beide Begriffe immer wieder bei der Beschreibung von Web-Applikationen auf.

In dieser Tradition der ständigen Verbesserungen ist sicherlich auch Struts zu sehen. Dieses soll zeigen, wie auf Grundlage von Servlets und JSP Web-Applikationen erstellt werden können, die oben beschriebene Mängel vermeiden und beide Techniken zu einem sinnvollen Ganzen zusammenfügen.

[1] Der Name begründet sich auf den Konferenzraum, in dem die Apache Group und Sun ihre Zusammenarbeit beschlossen. Zweifellos wurde die Namensfindung durch die Tatsache erleichtert, dass Jakarta die auf der Insel Java liegende Hauptstadt Indonesiens ist [TSS01, Seite 346].

Hierbei ist mit dem Tomcat-Server ein weiteres Jakarta-Projekt häufig in Zusammenhang mit Struts anzutreffen. Tomcat stellt die Referenzimplementierung von Sun für die beiden Techniken Servlets und JSP, also genau die von Struts verwendeten, dar. Innerhalb der J2EE-Spezifikation (s. Seite 20) stellt Tomcat also den Web-Container bereit, also die Laufzeitumgebung von Web-Applikationen, die mit Struts entwickelt werden.

5.1.2 Zielsetzung

Struts verbindet mit seiner Umsetzung der MVC-Architektur verschiedene Ziele. Eines von diesen ist die aufgrund der Architektur entstehende lose Koppelung der in den Bestandteilen Modell, View und Controller verwendeten Klassen. Eine lose Koppelung von Klassen erhöht immer ihre Wiederverwendbarkeit, da hierdurch Abhängigkeiten unter ihnen so gering wie möglich gehalten werden.

Zudem unterstützt Struts in den Views die Internationalisierung der Applikation und bietet in diesen auch einen Mechanismus zur Anzeige von Fehlermeldungen an. Für die Integration von im Framework verwendeten Klassen in die Views stellt Struts verschiedene Techniken zur Verfügung.

Zielsetzung für den Controller ist es, diesen leicht konfigurierbar zu machen, so dass Änderungen im Kontrollfluss der Applikation rein deklarativ vorgenommen werden können. Als Folge hiervon stellt der Controller zudem den zentralen Einstiegspunkt für die Applikation dar, an den alle eingehenden Anfragen gerichtet und beantwortet werden.

Über das Modell macht Struts lediglich geringe Aussagen, sein Schwerpunkt liegt eindeutig in den Controller- und View-Komponenten. Das Framework ist allerdings so gestaltet, dass es mit unterschiedlichen Modell-Techniken, wie Datenbanken oder J2EE, eingesetzt werden kann, die leicht in dieses integriert werden können.

5.1.3 Verbreitung

Das Struts Framework wird seit 1999 entwickelt, die aktuelle Version ist 1.0.2. Es war von Anfang an als Open-Source verfügbar, was sicherlich zu seiner weiten Verbreitung beigetragen hat. So sind im Internet eine Vielzahl an Informationen zu Struts auch ausserhalb seiner Homepage *http://jakarta.apache.org/struts/* anzutreffen. Sehr umfangreiche Informationen finden sich in zwei Mailinglisten unter *http://www.mail-archive.com* mit täglich bis zu 100 Beiträgen und unter *news.basebeans.com* ist eine eigene Newsgroup für Struts zu finden, die ebenfalls rege besucht wird.

In dieser Arbeit stütze ich mich überwiegend auf die über die Homepage verfügbare Dokumentation sowie die von Turau in [TSS01] sehr detaillierte Untersuchung des Frameworks. Diese bezieht sich zwar auf eine ältere Version von Struts, der grundlegende Aufbau des Frameworks hat sich allerdings nicht geändert.

5.2 Struts Umsetzung der MVC-Architektur

Im Folgenden beziehe ich mich überwiegend auf die von Turau in [TSS01, Seite 145] vorgenommene Unterteilung der von Struts verwendeten MVC-Architektur.

5.2.1 Aufbau

Die Umsetzung der MVC-Architektur in Struts entspricht dem auf Seite 35 beschriebenen Vorgehen *Vermittler-Servlet an Worker-Bean*. Hierfür nutzt Struts einen technischen Controller als zentrales Element. Er nimmt alle Anfragen (HTTP-Requests) des Benutzers an die Web-Applikation entgegen. Sind in diesen Parameter, wie z.B. jene aus HTML-Formularen, enthalten, so speichert er sie in speziellen Beans des Front-End-Modells ab. Danach wählt er anhand einer Konfigurationsdatei (*struts-config.xml*) den für die Anfrage zuständigen fachlichen Controller aus.

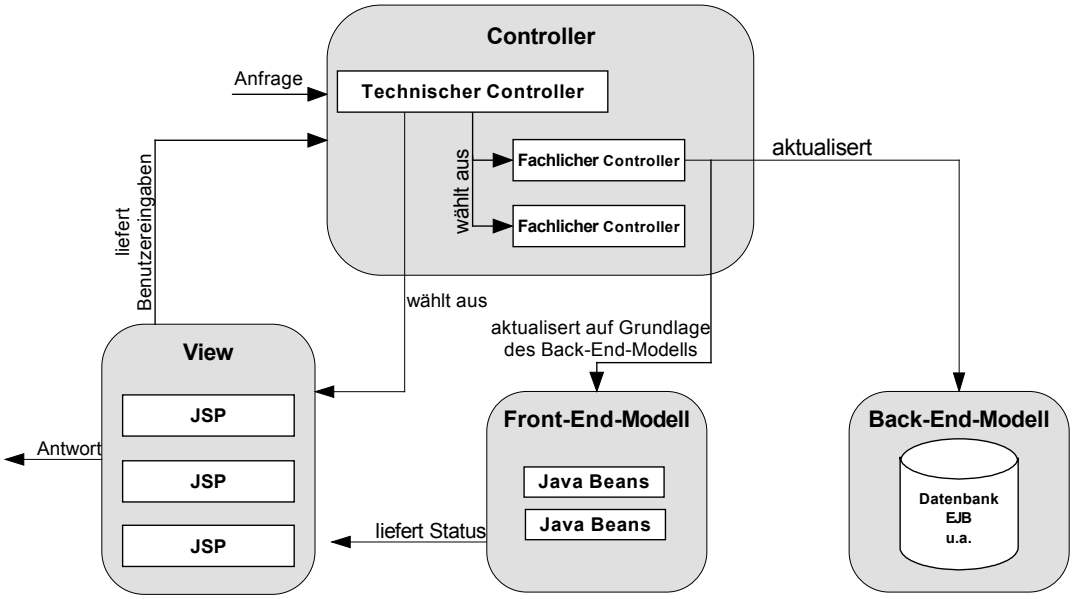

Abbildung 5.1: *Umsetzung der MVC-Architektur in Struts*

Dieser entscheidet nun auf Grundlage des Front-End-Modells sowie der Anfrage, welche Geschäftslogik im Back-End-Modell ausgeführt werden soll. In diesem sind die eigentliche Geschäftslogik sowie die Geschäftsdaten enthalten, z.B. in Form einer Datenbank oder in EJB. Aufgrund der Ergebnisse der Geschäftslogik aktualisiert der fachliche Controller nun das Front-End-Modell, welches den von den Views benötigten Ausschnitt aus den Geschäftsdaten enthält. Danach gibt er die Kontrolle zurück an den technischen Controller. Dieser entscheidet nun, wiederum auf Grundlage der Konfigurationsdatei, welcher View zur Anzeige verwendet werden soll. Dieser View, als JSP vorliegend, kann nun die zur Darstellung notwendigen Daten aus dem Front-End-Modell lesen und in einer HTML-

Seite verwenden. Diese wird abschließend an den Benutzer gesendet und kann nun wiederum Formulare enthalten, die im nächsten Schritt wieder an den technischen Controller gesendet werden.

5.2.2 Der Controller

Der technische Controller

Der technische Controller wird innerhalb des Frameworks durch das ActionServlet umgesetzt. Dieses ist fester Bestandteil des Frameworks und muss vom Entwickler lediglich konfiguriert werden. Dies geschieht über die XML-Datei *struts-config.xml*. Der technische Controller hat drei Aufgaben zu erfüllen:

1. Er dient als Schnittstelle zwischen HTTP-Anfragen und Framework.
2. Er kontrolliert den Workflow innerhalb des Frameworks.
3. Er wählt den nächsten anzuzeigenden View aus.

Die einzelnen Aufgaben sollen nun näher betrachtet werden.

Schnittstelle zwischen HTTP-Anfragen und Framework

Die Parameter eingehender HTTP-Requests werden vom technischen Controller automatisch in ActionForms gespeichert. Diese ActionForms enthalten für jeden Parameter ein gleichnamiges Attribut sowie die zu deren Lesen und Schreiben notwendigen set- und get-Methoden. Anhand der vom Request nachgefragten URI und der in der Konfigurationsdatei hinterlegten Zuordnung erkennt das ActionServlet, welche ActionForm für welche Anfrage zu verwenden ist. Durch diese Vorgehensweise muss sich der Entwickler also nicht mehr mit einzelnen HTTP-Parametern beschäftigen. Vielmehr fungiert hierbei das ActionServlet als Schnittstelle zwischen dem HTTP-Protokoll und der für den Entwickler leichter handhabbaren objektorientierten Umgebung.

Kontrolle des Workflows innerhalb des Frameworks

Nächster Schritt ist die Auswahl der auszuführenden Applikationslogik. Auch hier greift das ActionServlet wieder auf die Konfigurations-Datei zu und erhält von dort den für die Anfrage zuständigen fachlichen Controller. Diesem fachlichen Controller, im Framework als Action-Klasse implementiert, wird nun in der Methode *perform* die oben erzeugte ActionForm übergeben.

Auswahl des Views

Hat der fachliche Controller, also die jeweilige Action-Klasse, die Arbeit beendet, gibt die *perform*-Methode dem ActionServlet ein ActionForward-Objekt zurück. Anhand dieses Objekts sowie der Konfigurationsdatei kann das ActionServlet nun den anzuzeigenden View bestimmen, der dann an den Client gesendet wird.

Ein Blick in die Konfigurationsdatei soll die Zusammenhänge nochmals verdeutlichen:

```
...
<form-beans>
  <form-bean name="logonForm"
          type="niebel.forms.LogonForm">
  </form-bean>
</form-beans>
<global-forwards>
  <forward name="error" path="/error.jsp"/>
</global-forwards>
<action-mappings>
  <action path="/login"
        name="logonForm"
        input="/logon.jsp"
        type="niebel.actions.Login"
        validate="true">
        <forward name="success" path="/startseite.jsp"/>
        <forward name="failure" path="/logon.jsp"/>
  </action>
</action-mappings>
...
```

Bei einer Anfrage an die Adresse *./login.do* prüft das ActionServlet aufgrund des name-Tags, ob in der Session unter *logonForm* eine ActionForm vorhanden ist. Falls nicht, legt es ein entsprechendes Objekt an, die Klasse bestimmt es dazu aus dem entsprechenden *form-bean*-Eintrag. Die mit der Anfrage eingegangenen Parameter werden nun in die LogonForm eingetragen. Da *validate* auf *true* gesetzt ist, wird die gleichnamige Methode der ActionForm aufgerufen. Sollte diese ein nicht leeres Fehler-Objekt zurückgeben,weil zum Beispiel gar kein Passwort angegeben wurde, wird die unter *input* angegebene Seite als View ausgewählt. Ansonsten instanziiert das ActionServlet ein Objekt der Klasse *niebel.actions.Login* und übergibt diesem die LogonForm. Den von diesem Objekt zurückgegebenen ActionForward vergleicht das ActionServlet zunächst mit den lokalen Werten *success* und *failure*. Entspricht er keinem von diesen, werden die *global-forwards* durchsucht und die entsprechende Antwortseite zurückgegeben.

Der fachliche Contoller

Der fachliche Controller wird innerhalb des Frameworks durch die Action-Klassen dargestellt. Sie müssen vom Entwickler von der Klasse *org.apache.struts.action.Action* abgeleitet und implementiert werden. Der fachliche Controller hat folgende Aufgaben zu erfüllen:

1. Das Ausführen der eigentlichen Geschäftlogik.
2. Die Aktualisierung des Front-End-Modells.
3. Die Rückgabe eines ActionForwards an den technischen Controller.

Ausführen der Geschäftslogik

Dem jeweiligen fachlichen Controller, also einer Action-Klasse, wird beim Aufruf der *perform*-Methode eine ActionForm übergeben. Anhand der get-Methoden dieser Klasse kann er die Eingaben des Benutzers abfragen und den vom ihm betätigten Button erkennen. Auf Basis dieser Informationen kann er nun entscheiden, welche Geschäftslogik auszuführen ist. Bei kleineren System kann die Geschäftslogik direkt in der Action-Klasse implementiert werden. Grundsätzlich empfiehlt es sich jedoch, diese in eine eigene Schicht auszulagern, da hierdurch die Wiederverwendbarkeit der Action-Klassen gesteigert wird. Die einzelnen Möglichkeiten werden bei der Beschreibung des Modells näher erläutert.

Aktualisierung des Front-End-Modells

Nach Ausführung der Geschäftslogik muss die Action-Klasse das von den Views verwendete Front-End-Modell aktualisieren. Meist sind hier lediglich die Attribute der jeweiligen ActionForm über die entsprechenden set-Methoden zu ändern. Stellenweise wird dieser Teil des Modells aber auch durch in der Session gespeicherte FrontEndModellBeans repräsentiert, so dass auch diese gegebenenfalls zu aktualisieren sind.

Rückgabe eines ActionForwards

Abschließend muss die Action-Klasse anhand des Ergebnisses aus der Geschäftslogik entscheiden, welcher ActionForward an den technischen Controller zurückgegeben werden soll. Hierzu verwendet es symbolische Konstanten, die in der Konfigurationsdatei hinterlegt sind. Diese Konstanten existieren auf zwei Ebenen: lokal und global. Globale ActionForwards können von allen Actions verwendet werden, während lokale nur von der jeweiligen Action verwendet werden können. Hierbei ist zu beachten, dass die lokalen Bezeichner immer zuerst ausgewertet werden. Nur wenn sich der entsprechende Wert hier nicht findet, wird auf globaler Ebene danach gesucht. Nach Rückgabe des ActionForwards geht die Kontrolle zurück an den technischen Controller.

5.2.3 Das Modell

Wie schon erwähnt, unterteilt Struts das Modell in einen Front-End- und einen Back-End-Bereich. Während das Front-End-Modell sehr nahe bei den Views einzuordnen ist, enthält das Back-End-Modell die eigentlichen Geschäftsdaten und -logik.

Das Front-End-Modell

Das Front-End-Modell hat drei Aufgaben zu erfüllen:

1. Es dient als Verbindung zwischen dem Framework und den ein- und ausgehenden HTTP-Meldungen.

2. Es muss den Views den aktuellen von ihnen verwendeten Ausschnitt aus den Geschäftsdaten zur Verfügung stellen.

3. Es beinhaltet den aktuellen Zustand der Applikation.

Die ersten beiden Anforderungen werden über die ActionForms erfüllt. Vereinfacht kann zunächst angenommen werden, dass für jedes in den Views enthaltene Formular vom Entwickler eine ActionForm erstellt werden muss. Diese enthält für jedes Formularfeld ein als *private* deklariertes Attribut, auf das über set- und get-Methoden zugegriffen werden kann. Zudem sollte eine reset-Methode definiert werden, welche die Attribute auf Standardwerte zurücksetzt. Ebenfalls optional, aber zu empfehlen, ist das Überschreiben der validate-Methode der Action-Form-Klasse. In ihr sollte eine erste Überprüfung der Attribute vorgenommen werden.

Zur Speicherung des Zustands der Applikation sind die ActionForms meistens ausreichend. Zusätzliche Informationen können in den frei gestaltbaren FrontEndModellBeans implementiert werden. Diese können zum Beispiel genutzt werden, um mehrere ActionForms zu verwalten oder Beziehungen zwischen diesen zu modellieren. Sie werden dann gewöhnlich in der Session abgespeichert.

Das Back-End-Modell

Das Back-End-Modell stellt das eigentliche MVC-basierte Modell in Struts dar. Seine Aufgaben bestehen in der Modellierung der der Applikation zugrunde liegenden Geschäftsdaten sowie in der Bereitstellung einer auf diesen Daten ablaufenden Geschäftslogik. Im Gegensatz zum Front-End-Modell, dessen Erstellung nur geringe Alternativen zulässt, sind die Möglichkeiten für die Erstellung des Back-End-Modells sehr vielfältig. Zum einen empfiehlt Struts verschiedene Bean-Arten für die Implementierung des Back-End-Modells, zum anderen können diese unterschiedlich eingesetzt und kombiniert werden.

Im einzelnen schlägt Struts die Benutzung der folgenden Bean-Arten vor:

System State Beans: Die Apache-Group versteht hierunter Beans, die den aktuellen Zustand des Systems repräsentieren. Bei kleineren System ist es denkbar, dass der gesamte Zustand des Systems in System State Beans festgehalten wird. Im Normalfall repräsentieren sie jedoch Daten, die in einen externen Datenbank abgespeichert sind.

Business Logic Beans: Hierbei handelt sich um Beans, in deren Methoden die Geschäftslogik festgehalten wird.

Actions: Die Actions sind von Struts definierte Klassen, die zwar eigentlich den fachlichen Controller implementieren, die jedoch zur Erstellung der Geschäftslogik genutzt werden können.

Mit Hilfe dieser Bean-Arten können nun Geschäftsdaten und Logik unterschiedlich implementiert werden:

Logik in Action-Klassen: Hierbei wird die Geschäftslogik in den Action-Klassen implementiert, während der Zustand in System State Beans enthalten ist. Diese Vorgehensweise ist nur für kleine Projekte empfehlenswert, da die Action-Klassen den fachlichen Controller darstellen und es so zu einer Aufweichung des MVC-Musters kommt. Im Ergebnis erhält man schwer wartbare Action-Klassen, die durch die Implementierung der Geschäftslogik zudem die gesamte Applikation sehr unflexibel werden lassen.

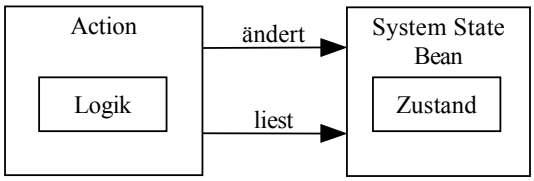

Abbildung 5.2: *Logik in Action-Klassen*

Logik in System State Beans: Bei dieser Vorgehensweise werden sowohl Zustand als auch Logik in denselben Klassen implementiert. Durch diese Vorgehensweise entsteht eine deutliche Trennung des Back-End-Modells von der restlichen Applikation. Das Back-End-Modell liegt nun als echte Schicht innerhalb des Systems vor. Nachteile liegen jedoch in der Verschmelzung von Logik und Zustand, sie sollten nicht unterschätzt werden.

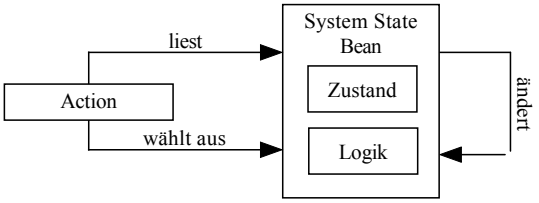

Abbildung 5.3: *Logik in System State Beans*

Logik getrennt: Die dritte Möglichkeit besteht nun in der strikten Trennung von Zustand und Logik in unterschiedlichen Beans: Während der Zustand in System State Beans festgehalten wird, wird die Logik in Business Logic Beans erstellt. Diese Vorgehensweise ist zwar die aufwändigste, sie bietet jedoch auch die größtmögliche Flexibilität. Durch die feinere Granularität der verwendeten Beans erhöht sich zudem die Wahrscheinlichkeit ihrer Wiederverwendbarkeit.

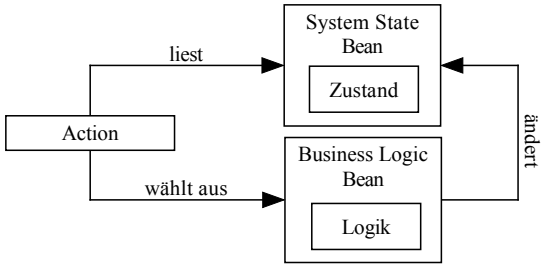

Abbildung 5.4: *Logik getrennt*

5.2.4 Die Views

Die Views haben folgende Aufgaben zu erfüllen:

1. Die Darstellung des Front-End-Modells.
2. Internationalisierung der Applikation.
3. Die Darstellung von Fehlermeldungen.

Darstellung des Front-End-Modells
Wie bei der Beschreibung des Modells erläutert, wird dass Front-End-Modell durch ActionForms sowie zusätzliche Java-Beans repräsentiert. Um diese Klassen nun in den Views darstellen zu können, sind in Struts mehrere Tag-Bibliotheken enthalten. Mit ihnen ist es möglich, auch komplizierte Darstellungen in den Views lediglich auf Basis von Tags vornehmen zu können.

Zur Darstellung des Front-End-Modells stellt Struts eine mächtige Tag-Bibliothek zu Verfügung. Sie ist einerseits ausreichend, um auch umfangreichere Aufgabenstellungen erfüllen zu können, anderseits aber auch leicht zu verstehen und somit von den Seiten-Gestaltern leicht anzuwenden. Die Tag-Bibliothek unterteilt sich in die folgenden Bereiche:

- Bean-Tags dienen der leichteren Einbindung der ActionForms in die Views.
- HTML-Tags bieten Unterstützung bei der Erstellung der Views, insbesondere beim Erstellen der hierin verwendeten Formulare. Über die Konfigurationsdatei können den Views ActionForms zugeordnet werden. Die HTML-Tags erlauben es nun, die jeweiligen Attribute der ActionForms einfach mit den Eingabeelementen des Formulars, wie z.B. Textfeldern oder Auswahllisten zu verknüpfen. Die Tags garantieren nun, dass bei einer Übermittlung der Formulardaten deren Werte in den entsprechenden Attributen der ActionForm gespeichert werden.
- Die Logic-Tags erlauben das Erstellen von Präsentationslogik, ohne hierbei Java-Code verwenden zu müssen. Sie unterstützen somit insbesondere die Trennung von Design und Programmierung. Mit ihnen ist es auch möglich, über ActionForms oder Teile von diesen zu iterieren.

Internationalisierung der Applikation

Ein weitere Aufgabe der Views ist die Internationalisierung der Anwendung. Hierzu können so genannte Ressource-Dateien für die zu verwendenden Sprachen angelegt werden. Sie enthalten jeweils einen Schlüssel und die Übersetzung in der entsprechenden Sprache. In den Views wird nun lediglich auf die Schlüssel zugegriffen, die Auswahl der Übersetzung fällt auf Grundlage der aktuellen Spracheinstellung des Benutzers. Diese kann automatisch über die HTTP-Header oder manuell durch den Benutzer erfolgen.

Darstellung von Fehlermeldungen

Sowohl in den Actions als auch in den ActionForms können Fehlermeldungen erzeugt werden. Diese werden in ActionError-Objekten gespeichert und in der Session abgelegt. Um auch hierbei die Internationalisierung zu unterstützen, werden die Meldungen ebenfalls in der Ressource-Datei der Applikation abgelegt. Innerhalb des Frameworks werden dann lediglich die entsprechenden Schlüsseln verwendet.

In den Views können diese Fehlermeldungen nun über spezielle Tags abgefragt und angezeigt werden.

5.2.5 Zusammenfassung

Die Verwendung der einzelnen Klassen soll nochmals analog zu Abbildung 5.1 verdeutlicht werden.

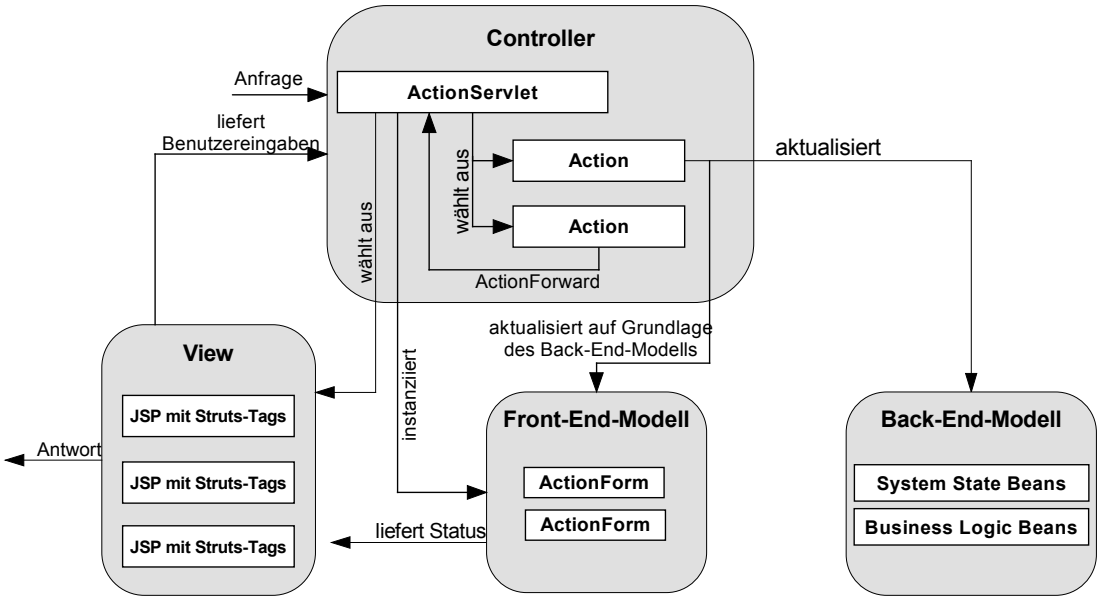

Abbildung 5.5: *Umsetzung der MVC-Architektur in Struts*

Der Controller setzt sich aus dem in der *struts-config.xml* konfigurierten ActionServlet und den vom Entwickler zu erstellenden Action-Klassen zusammen. Für das Back-End-Modell sind System State sowie Business Logic Beans zu erstellen, während für das Front-End-Modell die ActionForms implementiert werden müssen. Abschliessend sind die Views unter Verwendung der Struts-Tags zu erstellen.

Sämtliche an die Applikation gerichtete Anfragen werden vom ActionServlet entgegengenommen. Dieses wählt nun aufgrund der *struts-config.xml* die auszuführende Action aus und instanziiert zuvor eine eventuell benötigte ActionForm, die es mit den in der Anfrage enthaltenen Parametern belegt. Die Action wählt nun zunächst auf Grundlage der Anfrage die auszuführenden Business Logic Beans aus. Danach aktualisiert sie an Hand der System State Beans das Front-End-Modell, also die ActionForms. Anschliessend übergibt die Action einen ActionForward an das ActionServlet. Dieses entscheidet nun wiederum aufgrund der *struts-config.xml*, welcher View angezeigt werden soll. Der View greift dann über die Struts-Tags auf das Front-End-Modell zu und wird als Antwort zum Benutzer gesendet, wo er erneut Benutzereingaben aufnehmen und an die Applikation zurücksenden kann.

5.3 Erstellen einer Web-Applikation mit Struts

Im Folgenden soll mit Struts eine kleine Web-Applikation erstellt werden. Es wird die Erstellung eines Teiles dieser Applikation vorgestellt, die gesamte Applikation befindet sich auf der CD. In Kapitel 6 wird zum Vergleich gezeigt, wie sich ein weiterer Bestandteil der Applikation mit dem Oracle MVC Framework erstellen lässt.

5.3.1 Beschreibung der Web-Applikation

Es ergab sich hierfür die Erstellung einer Web-Applikation im Umfeld der Fachhochschule, deren Anforderungen kurz beschrieben werden sollen:

Es existieren im Fachbereich Informatik zwei Arten von Fächern: Pflichtfächer und Wahlpflichtfächer. Pflichtfächer werden vom Dekanat im Lehrplan vorgeschrieben. Das Dekanat legt hierfür auch die Dozenten fest die lediglich die Inhalte der Pflichtfächer in vorgegeben Rahmen, bestimmen können. Dem gegenüber können Wahlpflichtfächer von den Dozenten in Eigenverantwortung angeboten werden.

Unzufriedenstellend ist nun der Umstand, dass die Information welche Dozenten welche Wahlpflichtfächer anbieten, nicht an einer zentralen Stelle verwaltet werden. Sie sind vielmehr auf Aushängen oder den jeweiligen Internet-Seiten der Dozenten verstreut anzutreffen. Das Ziel soll also sein, dem Dekanat und den Dozenten eine Applikation zur Verfügung zu stellen, mit der die Fächer von dem jeweils Verantwortlichen verwaltet werden können. Als Datenbank soll hierzu das Hochschul-Informations-System (HIS) dienen, eine im Rahmen einer Projektarbeit erstellte Oracle-Datenbank.

Aus dieser Problemstellung heraus ergeben sich folgende Anforderungen an die Web-Applikation:

Die Benutzung der Applikation erfolgt über einen Web-Browser. Zur Benutzung ist die Eingabe eines Benutzernamens und Passworts notwendig.

Das Dekanat kann Dozenten anlegen und löschen.

Das Dekanat kann Pflichtfächer anlegen,ändern und löschen. Ihnen müssen ein oder mehrere Dozenten zugewiesen werden. Ebenfalls müssen den Pflichtfächern bei der Anlage Studiengang und Semester zugewiesen werden.

Dozenten können lediglich die Inhalte der ihnen vom Dekanat zugewiesenen Pflichtfächern ändern.

Dozenten können Wahlpflichtfächer anlegen, ändern und löschen. Bei der Anlage müssen diesen Studiengang und Semester zugewiesen werden. Es können zudem jederzeit zusätzliche Dozenten für ein Wahlpflichtfach angegeben werden.

Sind für ein Wahlpflichtfach mehrere Dozenten eingetragen, so sind diese in der Verwaltung des Wahlpflichtfachs gleichberechtigt.

> Zur Datenspeicherung wird die Datenbank des Hochschul-Informations-Systems (HIS) benutzt, Zugriffsmöglichkeiten bestehen über das Intranet der Fachhochschule bzw. über das Internet.

Eine erste Überlegung zeigt, dass zur Erfüllung der Anforderungen zwei Applikationen notwendig sind: HISDekanat und HISDozent. HISDekanat soll es dem Dekanat ermöglichen, Dozenten und Pflichtfächer nach den oben angebenen Anforderungen zu verwalten. Ergänzend hierzu soll HISDozent den Dozenten die Verwaltung der Wahlpflichtfächer sowie das Ändern der Pflichtfachinhalte ermöglichen.

Bei der Erörterung des Struts-Frameworks werde ich die Entwicklung von HISDozent erläutern, während ich beim Oracle MVC Framework auf die Erstellung von HISDekanat eingehe. Beide Applikationen habe ich im Rahmen dieser Arbeit mit Struts vollständig erstellt. Sie sind inklusive der HIS-Datenbank auf der beigefügten CD enthalten.

5.3.2 Installation des Tomcat-Servers und Struts

Zum Arbeiten mit Struts empfiehlt sich der Einsatz des Tomcat-Servers. Er unterstützt mit JSP und Servlets die von Struts benötigten Techniken und da er hierfür die Referenzimplementierung ist, sind auf ihm entwickelte Applikationen kompatibel zu weiteren Servern.

Tomcat kann unter *http://jakarta.apache.org/tomcat* heruntergeladen werden und benötigt zusätzlich ein Java Development Kit ab Version 1.2, welches unter der Umgebungsvariable *javahome* erreichbar sein muss.

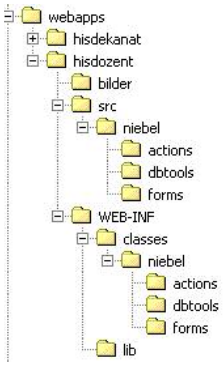

Abbildung 5.6: *Verzeichnisstruktur der Web-Applikationen in Tomcat*

Die Web-Applikationen werden nun im Verzeichnis *webapps* angelegt. Für jede Applikation ist ein separates Verzeichnis anzulegen. Lokal kann über *http://127.0.0.1:8080/Applikationsname* auf sie zugegriffen werden.

Um innerhalb der Applikation Struts verwenden zu können, werden die Struts-Bibliotheken im Verzeichnis */HISDozent/WEB-INF/lib* abgelegt. Parallel hierzu liegen im Ver-

zeichnis */HISDozent/WEB-INF/classes* die class-Dateien der für die Applikation erstellten Klassen. Während der Entwicklung sollten deren Quelltext-Dateien unter */HISDozent/src* abgespeichert werden. Die JSP werden im Verzeichnis */HISDozent* abgelegt.

Bevor mit der Entwicklung begonnen werden kann, ist die Konfigurationsdatei */HISDozent/WEB-INF/web.xml* anzupassen:

```
...
<servlet>
  <servlet-name>action</servlet-name>
    <servlet-class>
      org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>HISDozent</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
...
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
...
```

Hierbei wird zunächst das ActionServlet konfiguriert. Der Wert für *application* gibt unter anderem an, welche Ressource-Datei für die Internationalisierung und die Fehlermeldungen verwendet werden soll. Für die Applikation HISDozent muss diese dann als *HISDozent.properties* unter */hisdozent/WEB-INF/classes* angelegt werden. Der Parameter *config* gibt an, welche Konfigurationsdatei für das Struts-Framework verwendet werden soll. Abschliessend werden alle eingehenden Anfragen die mit*.*do* enden über das Servlet-Mapping an das ActionServlet weitergeleitet.

Da HISDozent auf die HIS-Datenbank zugreifen muss, wird in der Applikation ein Pool zum Verwalten der Datenbankverbindungen verwendet. Auch dieser wird in der *web.xml* konfiguriert:

```
<servlet>
  <servlet-name>pool</servlet-name>
    <servlet-class>his.db.pool</servlet-class>
      <init-param>
        <param-name>dbdriver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
       </init-param>
```

```
    <init-param>
      <param-name>dburl</param-name>
      <param-value>
       jdbc:oracle:thin:his/hisuser@localhost:1521:dbfh
      </param-value>
    </init-param>
    <init-param>
      <param-name>dbmaxconnections</param-name>
      <param-value>20</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
```

Der Tomcat-Server ist nun ausreichend konfiguriert und es kann mit dem Erstellen von HISDozent begonnen werden.

5.3.3 Erstellen von HISDozent

Als erster Schritt empfiehlt sich die Erstellung einer Skizze, in der die gesamte Applikation grob dargestellt ist. Für HISDozent sieht diese wie folgt aus:

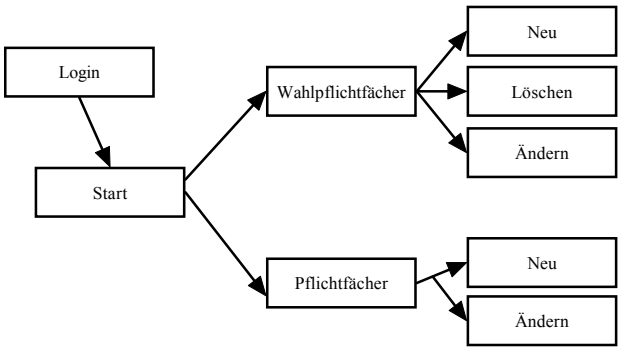

Abbildung 5.7: *Skizze für HISDozent*

Nach dem Login gelangt der Dozent auf eine Startseite, in der die Applikation kurz beschrieben wird. Von hier aus hat er nun die Möglichkeit, auf die Pflicht- bzw. Wahlpflichtfächer zuzugreifen und diese zu verwalten.

Im nächsten Schritt ist diese Skizze nun im Hinblick auf Struts zu verfeinern. Ziel ist es, die benötigten Actions, ActionForms, ActionForwards sowie JSP zu identifizieren und ihre Zusammenhänge zu verdeutlichen. Es empfiehlt sich, zunächst die benötigten JSP anzugeben. Werden von diesen ActionForms verwendet, so sollten auch diese mit eingetragen werden. Links aus den JSP sollten immer auf eine Action führen, dies gewährleistet, dass sämtliche Anfragen des Benutzers an das ActionServlet gerichtet und somit zentral konfiguriert werden können. Links von den JSP werden später in den Views durch Formulare oder HTML-Tags umgesetzt.

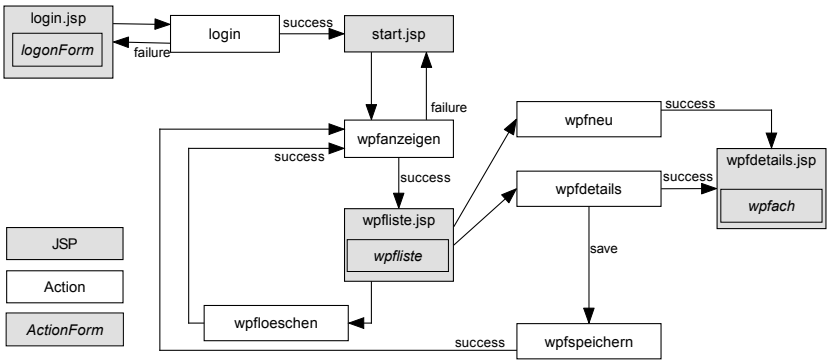

Abbildung 5.8: *Skizze für Wahlpflichtfächer*

Nachdem die benötigten Actions in die Skizze eingetragen wurden, müssen nun noch deren Verbindungen zu weiteren Actions bzw. den JSP festgelegt werden. Diese entsprechen den im Framework verwendeten ActionForwards und sind daher mit symbolischen Bezeichnern zu versehen. Als Ergebnis sollte man eine Skizze ähnlich zu Abbildung 5.8 erhalten.

Konfiguration des technischen Controllers

Ausgehend von dieser Skizze kann nun die Konfigurationsdatei *struts-config.xml* für den technischen Controller, also das ActionServlet, erstellt werden. Zunächst sind die beiden benötigten ActionForms einzutragen. Neben dem zu verwendenden Namen innerhalb der Applikation ist auch die zu benutzende Klasse anzugeben:

```
<form-beans type="org.apache.struts.action.ActionFormBean">
  <form-bean name="wpfach"
          type="niebel.forms.fach"/>
  <form-bean name="wpfliste"
          type="niebel.forms.fachliste"/>
</form-beans>
```

Im nächsten Schritt sind die verwendeten Actions anzugeben. Neben der Zuordnung eines Pfades und der Angabe einer Klasse muss hier auch der Name der ActionForm eingetragen werden, die von der Action bzw. der JSP verwendet werden soll. Über den Schalter *validate* kann festgelegt werden, ob die gleichnamige Methode der ActionForm vor dem Weiterleiten der Anfrage an die Action ausgeführt werden soll. Gibt diese eine Fehlermeldung zurück, so wird dem Benutzer der unter *input* festgelegte View angezeigt. Als Beispiel soll der Eintrag für die Action *wpfdetails* dienen:

```
<action path="/wpfdetails"
      type="niebel.actions.wpfdetails"
      name="wpfach"
      input="/wpfdetails.jsp"
      validate="true">
      <forward name="success"
         path="/wpfdetails.jsp"
         redirect="false"/>
```

```
      <forward name="save"
         path="/wpfspeichern.do"
         redirect="true"/>
</action>
```

Neben den schon erklärten Einträgen sind hier auch die ActionForwards zu sehen. In ihnen muss der Pfad angegeben werden, auf den im folgenden zugegriffen werden soll. Das Attribut *redirect* gibt an, ob die von der Action verwendete ActionForm ebenfalls weitergeleitet werden soll.

Nachdem die Einträge für alle Actions vorgenommen worden sind, ist das ActionServlet vollständig konfiguriert und es kann mit dem Erstellen der benötigten Klassen begonnen werden. Die Datei *struts-config.xml* ist auch im Anhang auf Seite III zu finden.

Erstellen des Front-End-Modells

Wie im vorigen Kapitel beschrieben, besteht das Front-End-Modell überwiegend aus den ActionForms. Für die Verwaltung der Wahlpflichtfächer werden ActionForms zur Anzeige der Liste sowie zur Anzeige von einzelnen Wahlpflichtfächern benötigt.

Alle verwendeten ActionForms sind von *org.apache.struts.action.ActionForm* abzuleiten. Für die benötigten Attribute sind set- und get-Methoden zu implementieren. Die ActionForm *fachliste* stellt sich somit wie folgt da:

```
package niebel.forms;
import org.apache.struts.action.ActionForm;
import java.util.*;
public class fachliste extends ActionForm
{
 private ArrayList Liste = new ArrayList();
 public Object getListe(int i){
  return Liste.get(i);
 }
 public ArrayList getListe() {
  return Liste;
 }
 public void setListe(ArrayList Liste) {
  this.Liste = Liste;
 }
}
```

Sie besitzt lediglich eine ArrayList in der später die Fächer gespeichert werden. Sie ist so allgemein wie möglich gehalten, um mit verschiedenen Klassen benutzt werden zu können. Um sie, wie später noch gezeigt wird, leicht in den Views darstellen zu können, besitzt sie zusätzlich eine Methode, die den Zugriff auf bestimmte Indizes der Liste ermöglicht. Eine Validierung ist innerhalb dieser ActionForm nicht notwendig.

Deutlich umfangreicher gestaltet sich die Erstellung der ActionForm *fach*, die zur Verwendung und Darstellung von Pflicht- und Wahlpflichtfächer dient. Ein Blick auf die

später in der Applikation verwendete Eingabemaske in Abbildung 5.9 soll zunächst einen Überblick geben:

Abbildung 5.9: *Eingabemaske für ein Wahlpflichtfach*

Die Werte für die Bezeichnung, das Kürzel sowie den Inhalt des Fachs können in Textfeldern eingegeben werden, weshalb sie in der ActionForm als Strings realisiert werden können. Für die Angabe der Prüfungsart, der Dozenten sowie der Studiengänge werden Auswahllisten benötigt, weshalb sie als ArrayList zu implementieren sind. Da sowohl mehrere Dozenten und Studiengänge für ein Fach ausgewählt werden können benötigt man zusätzliche ArrayLists, in denen die ausgewählten Dozenten und Studiengänge abgespeichert werden. Für all diese Attribute sind nun zunächst wieder die set- und get-Methoden zu erstellen.

Um den View möglichst komfortabel gestalten zu können, werden zudem Methoden zum Löschen der Listen für die ausgewählten Dozenten und Studiengänge hinzugefügt. Soll ein Fach gespeichert werden, ist zudem eine erste Überprüfung der eingegebenen Daten innerhalb der ActionForm möglich. Diese wird in der Methode *validate* vorgenommen:

```
public ActionErrors validate(ActionMapping actionMapping,
                      HttpServletRequest request)
{
  ActionErrors errors=new ActionErrors();
  if ("Speichern".equals(request.getParameter("action")))
  {
    if ("".equals(this.getKommentar().trim())){
      errors.add(ActionErrors.GLOBAL_ERROR,
        new ActionError("error.fach.keinkommentar" ));
  }
  if (dozenten.size()==0){
    errors.add(ActionErrors.GLOBAL_ERROR,
      new ActionError( "error.fach.keinedozenten" ));
  }
...
}
```

```
return errors;
}
```

Um die Überprüfung nur beim Speichern vorzunehmen, wird zunächst der Parameter *action* überprüft. Dieser enthält den Wert des im Formular gedrückten Buttons. Anschliessend werden nun die einzelnen Bedingungen, wie z.B. Vorhandensein eines Kommentars oder Angabe mindestens eines Dozenten, überprüft. Ist eine der Bedingungen erfüllt, wird ein ActionError erzeugt und dem *errors*-Objekt hinzugefügt, welches am Ende der Methode zurückgegeben wird. Sind Fehler aufgetreten, wählt das ActionServlet den in der *struts-config.xml* unter *input* festgelegten View als nächsten aus.

Die bei der Erzeugung der ActionError verwendeten Schlüssel müssen in der Ressource-Datei vorhanden sein und mit dem gewünschten Text belegt werden. Hierin kann zur Formatierung auch HTML verwendet werden. Für die beiden oben angegebenen Fehlermeldungen sehen die Einträge wie folgt aus:

```
error.fach.keinkommentar=
  <li>Bitte geben Sie den Inhalt des Fachs an.
error.fach.keinedozenten=
   <li>Bitte waehlen Sie mindestens einen Dozenten aus.
```

Auf die Darstellung der Fehlermeldungen in den Views wird weiter unten eingegangen.

Mit dem Erstellen der beiden ActionForms *fach* und *fachliste* ist das Front-End-Modell für die Verwaltung der Wahlpflichtfächer vollständig. Die ActionForms für die restliche Applikation werden analog zu dem hier beschriebenen Vorgehen erstellt.

Erstellen des Back-End-Modells

Als nächstes empfiehlt sich sich nun das Erstellen des Back-End-Modells. Im Hinblick auf den vergleichsweise geringen Umfang von HISDozent bietet sich hierbei das auf Seite 59 beschriebene Vorgehen *Logik in System State Beans* an. Hierdurch wird das Back-End-Modell mit einem vertretbaren Aufwand in eine eigene Schicht ausgelagert und kann somit leichter an Änderungen in der Datenbank angepasst werden.

Um von den Action-Klassen auf das Back-End-Modell zugreifen zu können, erweist sich das Erstellen einer HandlerFactory nach dem Fabrik-Muster als sinnvoll. Immer wenn eine Action auf das Back-End-Modell zugreifen muss, erhält es von dieser Factory den entsprechenden Handler:

```
public class HandlerFactory{
  public static FachHandler getFachHandler(){
    FachHandler f=new DBHandler();
    return f;
  }
}
```

Der von der Factory zurückgegebene FachHandler implementiert das entsprechende Interface, in dem alle für das Lesen und Schreiben der Wahlpflichtfächer notwendigen Methoden enthalten sind:

```
import niebel.forms.*;
public interface FachHandler{
   public fachliste getWpfliste(String id);
   public fach getFach(String id);
   public fach getFach(); //Gibt leeres Fach üzurck
   public void saveFach(fach f);
   public void deleteFach(String id);
}
```

Bei diesem Vorgehen ist zu beachten, dass die Implementierung dieses Handlers, die Klasse DBHandler, das Package *niebel.forms.** importieren muss, da er auf die ActionForms *fach* und *fachliste* zugreifen soll. Hierdurch kann das Back-End-Modell nur noch in Verbindung mit Struts verwendet werden, ohne Struts kann es nicht eingesetzt werden. Da das Back-End-Modell von HISDozent jedoch nur aus einer Klasse (*DBHandler*) besteht, ist dieses Vorgehen hier vertretbar. Bei umfangreicheren Applikationen muss hier jedoch eine stärkere Trennung vorgenommen werden, um das dann aufwendiger erstellte Back-End-Modell auch in anderen Umgebungen verwenden zu können. Häufig wird beim Erstellen einer Web-Applikation auch schon ein Back-End-Modell vorliegen, so dass man dieses dann über eine feinere Schichtenbildung in die Anwendung integrieren muss.

Es fehlt nun nur noch die Implementierung der Klasse *DBHandler*. In ihr müssen die im Interface *FachHandler* festgelegten Methoden ausprogrammiert werden. Für HISDozent wurde hierfür JDBC benutzt, mit dem unter Verwendung eines Connection-Pools Fächer aus der Datenbank gelesen und in dieser gespeichert werden. Für das Verständnis von Struts ist dies nicht weiter von Bedeutung, weshalb an dieser Stelle nicht weiter ins Detail gegangen werden soll.

Erstellen der fachlichen Controller

Nach Erstellung des Front- und Back-End-Modells können nun die benötigten fachlichen Controller, die Action-Klassen, erstellt werden. Die zu erstellenden Klassen sind aus der *struts-config.xml* ersichtlich. Allen gemeinsam ist, dass sie von *org.apache.struts.action.Action* abzuleiten sind und jeweils die Methode *perform* zu überschreiben ist. Diese muss einen der ActionForwards zurückgeben, die für die Action in der *struts-config* angegeben worden sind.

Da jede Action entweder mit dem Front- oder mit dem Back-End-Modell interagiert, soll ihre Entwicklung im Folgenden an Hand dieser Eigenschaft erläutert werden:

Zugriff auf Back-End-Modell

Ein Beispiel einer Action, die nur auf das Back-End-Modell zugreift ist *wpflöschen* die anhand einer vom View erhaltenden ID das Back-End-Modell veranlasst, das entsprechende Wahlpflichtfach zu löschen. Ihre *perform*-Methode sieht wie folgt aus:

```
public ActionForward perform(ActionMapping mapping,
                     ActionForm form,
                     HttpServletRequest request,
                     HttpServletResponse response)
```

```
{
      FachHandler f=HandlerFactory.getFachHandler();
      f.deleteFach(request.getParameter("id"));
      return mapping.findForward("wpfanzeigen");
}
```

Zunächst wird der benötigte FachHandler erzeugt. Aus dem HTTPRequest wird dann die ID ausgelesen und der entsprechenden Methode des Handlers übergeben. Abschliessend wird der ActionForward über das ActionMapping gesucht und zurückgegeben.

Action-Klassen wie diese, die lediglich auf das Back-End-Modell zugreifen, können als sehr gut wiederverwendbar angesehen werden, da sie lediglich vom verwendeten Back-End-Modell abhängig sind.

Zugriff auf Front-End-Modell

Diese Actions verändern Teile des Front-End-Modells. Ein Beispiel hierfür ist die Klasse *wpfdetails*, die auf Eingaben der in Abbildung 5.9 angezeigten Eingabemaske reagiert. Ihre *perform*-Methode sieht wie folgt aus:

```
...
   ActionErrors errors=new ActionErrors();
   String forward = "success";
   fach fa=(fach) form;
   if (request.getParameter("adddozent")!=null)
     fa.copyDozent();
   else if (request.getParameter("deletedozenten")!=null)
     fa.deleteDozenten();
   ...
   else if ("Speichern".equals(request.getParameter("action"))){
     ArrayList a=fa.getDozenten();
     HttpSession s=request.getSession();
     dozent d=(dozent) s.getAttribute("dozent");
     boolean gefunden=false;
     for (int i=0;i<a.size();i++)
       if (((dozent) a.get(i)).getKuerzel().equals(d.getKuerzel()))
         gefunden=true;
       if (gefunden){
       forward="save";
     }
     else{
       errors.add(ActionErrors.GLOBAL_ERROR,
         new ActionError( "error.fach.nichtselbstdabei" ));
       saveErrors(request, errors);
       forward="failure";
     }
   }
   else if ("Abbrechen".equals(request.getParameter("action")))
     forward="wpfliste";
```

```
   return mapping.findForward(forward);
...
```

Der in Abbildung 5.9 mit „»“ beschriftete Button trägt den Namen *adddozent*. Wird er gedrückt, kann dies innerhalb der Action über die erste if-Abfrage erkannt werden. Da der in der SelectBox ausgewählte Dozent mit der Anfrage übertragen und vom ActionServlet vor Ausführen der Action in die ActionForm eingetragen wird, kann dieser durch Aufrufen der *copyDozent*-Methode in die Liste der ausgewählten Dozenten kopiert werden.

Entsprechend funktioniert das Löschen der Liste der ausgewählten Dozenten. Der Button *Liste löschen* erhält im View den Namen *deletedozenten*. In der Action kann somit das Betätigen des Buttons erkannt und die entsprechende Methode der ActionForm *fach* aufgerufen werden.

Im weiteren überprüft die *perform*-Methode, ob einer der Buttons *Speichern* oder *Abbrechen* betätigt wurde und ändert dementsprechend den zurückgegebenen ActionForward. Wurde der Button *speichern* betätigt wird zudem überprüft, ob sich der Dozent auch selber in die Dozentenliste eingetragen hat. Hierzu wird aus einem beim Login in der Session abgespeicherten *dozent*-Objekt das Kürzel ausgelesen und mit denen in der Liste der ausgewählten Dozenten verglichen. Ist es in dieser nicht enthalten, wird ein ActionError erzeugt und dem *errors*-Objekt hinzugefügt. Dieses wird dann über *saveErrors* im Request gespeichert und kann, wie später noch gezeigt wird, leicht in den Views ausgegeben werden. Dies ist ein Beispiel für eine Überprüfung, die nicht in die *validate*-Methode der ActionForm *fach* integriert werden kann, da diese Klasse keine Kenntnis darüber hat, von welchem Dozenten sie verwendet wird.

Actions, die ausschliesslich das Front-End-Modell ändern, sind sehr eng mit dem jeweiligen View verbunden. Dies ermöglicht einerseits das Erstellen funktionsreicher Views, andererseits verringert sich hierdurch aber auch deutlich die Wiederverwendbarkeit der Action-Klasse in anderen Applikationen.

Front-End-Modell nach Back-End-Modell

Diese Action-Klassen werden immer benötigt, wenn Daten aus dem Front-End-Modell abgespeichert werden sollen, wie es z.B. beim Abspeichern eines Wahlpflichtfaches der Fall ist. Als Beispiel soll die Action *wpfspeichern* dienen:

```
public ActionForward perform(ActionMapping mapping,
                      ActionForm form,
                      HttpServletRequest request,
                      HttpServletResponse response)
{
   String forward = "success";
   fach fa=(fach) form;
   FachHandler fh=HandlerFactory.getFachHandler();
   fh.saveFach(fa);
   return mapping.findForward(forward);
```

```
}
```

Die Methode konnte sehr einfach erstellt werden, da die ActionForm an das Back-End-Modell übergeben werden kann. Wie bei der Erstellung des Back-End-Modells beschrieben, ist dies bei umfangreicheren Applikationen nicht ratsam. Bei diesen sollte eine solche Bindung des Back-End-Modells an ein spezifisches Front-End-Modell unbedingt vermieden werden, um die Wiederverwendbarkeit der erstellten Komponenten zu erhöhen.

Back-End-Modell nach Front-End-Modell

Diese Actions werden benötigt, wenn Daten aus dem Back-End-Modell gelesen und in den Views angezeigt werden sollen. Ein Beispiel hierfür ist das Anzeigen der Liste der Wahlpflichtfächer eines Dozenten, für welches die Action *wpfanzeigen* erstellt wurde, deren *perform*-Methode im Folgenden angegeben ist:

```
public ActionForward perform(ActionMapping mapping,
                     ActionForm form,
                     HttpServletRequest request,
                     HttpServletResponse response)
{
  String forward = "success";
  HttpSession s=request.getSession();
  dozent d=(dozent) s.getAttribute("dozent");
  FachHandler f=HandlerFactory.getFachHandler();
  form= f.getWpfliste(d.getKuerzel());
  s.setAttribute("wpfliste",form);
  return mapping.findForward(forward);
}
```

Beim erfolgreichen Login wird ein Objekt *dozent* in der Session gespeichert, welches alle benötigten Informationen über den Dozenten enthält. Sollen nun dessen Wahlpflichtfächer angezeigt werden, so holt die Action zunächst dieses Objekt aus der Session und übergibt das Kürzel des Dozenten an das Back-End-Modell. Von diesem erhält es dann die zugehörige Fächerliste als ActionForm zurück und speichert diese unter dem in der *struts-config.xml* verwendeten Bezeichner in der Session ab.

Für diesen Typ von Actions gilt dasselbe wie für den vorangehend beschriebenen: Dadurch, dass sowohl auf Objekte des Front-End- als auch des Back-End-Modells zugegriffen wird, sind diese Actions nur sehr schlecht wiederverwendbar.

Erstellen der Views

Im letzten Schritt sind nun noch die Views zu erstellen. Zunächst soll die Seite *wpfanzeigen.jsp* betrachtet werden, in der die Wahlpflichtfächer eines Dozenten aufgelistet werden:

Der *struts-config.xml* kann entnommen werden, dass in der JSP eine ActionForm *wpfliste* vom Typ *fachliste* angezeigt werden muss. Diese kann nun mit unterschiedlichen Tags

Abbildung 5.10: *Wahlpflichtfächer eines Dozenten*

in die JSP eingebunden werden. Zur besseren Übersichtlichkeit sind im folgenden Ausschnitt der JSP HTML-Tags und Beschriftungen nicht enthalten:

```
...
<nested:root name="wpfliste">
  <nested:iterate property="liste">
    <form action="./wpfaendern.do" method="post">
      <input type=hidden name="id" value="<nested:write property="id"/>">
      <input type=submit name="action" value="L&#xF6;schen">
      <input type=submit name="action" value="&#xC4;ndern">
    </form>
    <nested:write property="kuerzel"/>
    <nested:write property="name"/>
    <nested:write property="pruefung"/>
    <nested:iterate property="studiengaenge">
      <nested:write property="studiengang"/>
      <nested:write property="semester"/><br>
    </nested:iterate>
    <nested:iterate property="dozenten">
      <nested:write property="anrede"/>
      <nested:write property="titel"/>
      <nested:write property="nachname"/><br>
    </nested:iterate>
    <nested:write property="kommentar"/>
  </nested:iterate>
</nested:root>
...
```

Die *nested*-Tags von Struts erlauben es, Objektstrukturen zu durchlaufen und die jeweiligen Inhalte auszugeben. Hierzu wird zunächst *wpfliste* über den *root*-Tag als Wurzelelement angegeben. Über den *iterate*-Tag kann nun die in der ActionForm enthaltene Liste durchlaufen werden. In jedem Durchlauf wird zunächst ein HTML-Formular erzeugt, die ID des Fachs als verstecktes Feld in die Seite geschrieben und die Buttons zum Löschen und Ändern des Fachs eingetragen. Nach dem Formular kann nun mit der Ausgabe der Fächer begonnen werden. Hierzu können die Attribute des jeweiligen Faches über die

write-Tags ausgegeben werden. Da ein Fach mehreren Studiengänge und Dozenten zugeordnet werden kann, müssen für deren Ausgaben wiederum die *iterate*-Tags benutzt werden.

An diesem Beispiel wird sehr schön die Mächtigkeit der *nested*-Tags deutlich. Mit den verschachtelten *iterate*-Tags steigt man jeweils in der Objektstruktur eine Ebene tiefer: Der erste *iterate*-Tag liefert die *fach*-Objekte aus der *fachliste*. Die *write*-Tags auf dieser Ebene beziehen sich dementsprechend auf das *fach*-Objekt. Der *iterate*-Tag für die Studiengänge durchläuft nun die gleichnamige ArrayList des *fach*-Objekts und die in ihm enthaltenen *write*-Tags beziehen sich auf die in dieser enthaltenen Objekte vom Typ *Studiengang*. Entsprechendes gilt für den *iterate*-Tag der Dozenten, dessen *write*-Tags sich auf die entsprechenden Objekte beziehen. Als Properties lassen sich alle Attribute der ActionForms verwenden, auf die mit get-Methoden zugegriffen werden kann.

Bei der Erstellung der in Abbildung 5.9 dargestellten *wpfdetails.jsp* können weitere Aspekte bei der Erstellung der Views betrachtet werden:

```
...
<nested:form action="wpfdetails.do" >
  <nested:select property="pruefungselected">
    <nested:options property="pruefungen"/>
  </nested:select>
  <nested:select name="wpfach" property="dozentselected">
    <nested:iterate property="adozenten">
      <option value="<nested:write property="id"/>">
        <nested:write property="nachname"/>
        <nested:write property="anrede"/>
        <nested:write property="titel"/>
      </option>
    </nested:iterate>
  </nested:select>
  <nested:submit property="adddozent" value=">>"/>
  <nested:equal name="wpfach" property="anzahldozenten" value="0">
     Sie haben noch keinen Dozenten ausgew&#xE4;hlt.
  </nested:equal>
  <nested:notEqual name="wpfach"
    property="anzahldozenten" value="0">
    <nested:iterate property="dozenten">
      <nested:write property="titel"/>
      <nested:write property="nachname"/><br>
    </nested:iterate>
    <nested:submit
      property="deletedozenten" value="Liste l&#xF6;schen"/>
   </nested:notEqual>
...
```

Zunächst wird die ListBox für die Prüfungsarten erstellt. Die Auswahlmöglichkeiten sind in der ArrayList *pruefungen* enthalten, das jeweils ausgewählte Element wird im Attribut

pruefungselected gespeichert. Danach wird die ListBox für die Dozenten erzeugt, was etwas aufwendiger zu realisieren ist, da neben deren Nachnamen auch Titel und Anrede angezeigt werden sollen. Anschliessend wird über die Tags *notEqual* bzw. *equal* geprüft, ob schon Dozenten ausgewählt wurden. Ist dies nicht der Fall, wird eine entsprechende Meldung angezeigt, ansonsten wird die Liste der ausgewählten Dozenten ausgegeben. So ist es sehr leicht möglich, über Tags Präsentationslogik in die Seiten einzubinden. Auch hierbei kann wieder auf die Verwendung von Java-Code verzichtet werden.

Die eventuell in den Actions oder den ActionForms erzeugten Fehlermeldungen sollen über der Eingabemaske angezeigt werden. In den Views ist hierzu lediglich eine Zeile notwendig:

```
<html:errors/>
```

Treten Fehler auf, so werden die entsprechenden Fehlermeldungen über diesen Tag in die Seite geschrieben. Verfügt die Applikation zudem über mehrere Ressource-Dateien, so

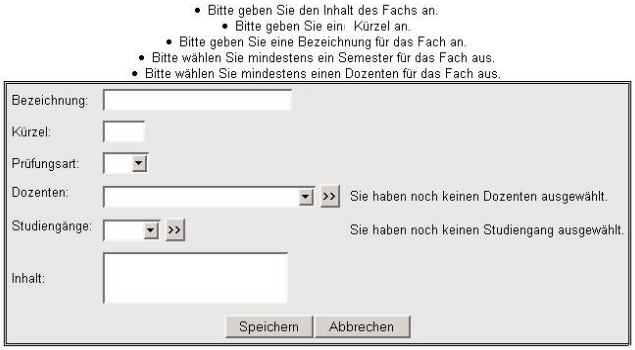

Abbildung 5.11: *Anzeige von Fehlermeldungen*

wählt der Tag die den Browser-Einstellungen entsprechende Sprache für die Anzeige aus. Die Buttons und Beschriftungen können ebenfalls für verschiedene Sprachen erstellt und in den Views angezeigt werden. Hierfür müssen auch in den Views Schlüssel verwendet werden, deren jeweilige Übersetzung in den entsprechenden Ressource-Dateien enthalten sein müssen. Sie können dann über den *message*-Tag in die Seiten eingebunden werden:

```
<html:message key="prompt.username"/>
```

Der Mehraufwand bei der Erstellung einer mehrsprachigen Applikation sollte hierbei nicht unterschätzt werden, da sämtliche in den Views und Klassen verwendeten Texte zu übersetzen sind. Sollte eine Internationalisierung der Anwendung allerdings notwendig sein, so bietet Struts alle hierfür notwendigen Vorraussetzungen.

Abschliessend werden nun die noch fehlenden Teile von HISDozent, wie z.B. das Ändern der Pflichtfächer, erstellt. Hierbei wird analog zu dem oben beschriebenen Verfahren vorgegangen.

Absichern der Applikation

Leider bietet Struts noch kein einheitliches Vorgehen für das Absichern der Web-Applikation an. Eine Möglichkeit besteht in der Erstellung einer Basis-Action in der die Berechtigung des Benutzers abgefragt wird. So wird bei HISDozent mit dem erfolgreichen Einloggen ein Objekt Dozent in der Session abgespeichert. In der Basis-Action wird nun zunächst die Existenz dieses Objektes geprüft. Ist es nicht vorhanden, wird dem Benutzer die Login-Seite angezeigt, anderenfalls wird die Methode *doPerform* ausgeführt:

```
...
public ActionForward perform(ActionMapping mapping,
                     ActionForm form,
                     HttpServletRequest request,
                     HttpServletResponse response)
      throws javax.servlet.ServletException, java.io.IOException
{
  HttpSession se=request.getSession();
  se=request.getSession();
  if(se.getAttribute("dozent")==null)
    return mapping.findForward("login");
  else
    return (doPerform(mapping,form,request,response));
}
...
```

Alle Action-Klassen die hierdurch geschützt werden sollen, sind nun von HISBaseAction abzuleiten und ihre *perform*-Methode ist in *doPerform* umzubenennen.

Tools bei der Erstellung

Für die Erstellung von HISDozent wurde lediglich eine Standard-Entwicklungsumgebung verwendet. Sollen umfangreichere Applikationen entwickelt werden, empfiehlt sich allerdings der Einsatz spezieller Tools. Unter *http://jakarta.apache.org/struts/resources/guis.html* befinden sich eine Liste verwendbarer Tools. Bei ihrem Einsatz sollte auf die von ihnen verwendete Struts-Version geachtet werden.

Hier findet man Tools, wie z.B. *StrutsBuilder* oder *Struts Console*, in denen zunächst über Eingabemasken Informationen zur Applikation eingegeben werden. Hieraus können dann die *struts-config.xml* sowie die Klassengerüste für die ActionForms und Actions generiert werden.

Einen Schritt weiter gehen Tools wie *Camino*, in denen eine zu Abbildung 5.8 vergleichbare Skizze erstellt werden kann. Hierdurch kann der Aufbau der Applikation noch besser verstanden und kontrolliert werden. Auf Grundlage dieser Skizze können dann wiederum alle benötigten Dateien generiert werden.

Erweiterung einer bestehenden Web-Applikation

Im Wintersemester 2001/2002 erstellte ich zusammen mit Michael Müller die Web-Applikation *Ausfallmeldesystem* (AMS). Diese sollte Dozenten dazu dienen, Ausfälle von

Vorlesungen und sonstige Mitteilungen über das Internet in das HIS eintragen zu können. Wir verwendeten auch hierfür Struts, jedoch sahen wir damals eine Erweiterung der Applikation nicht vor.

Es bot sich nun an, die in dieser Arbeit erstellte Applikation HISDozent in das AMS zu integrieren. Diese Integration erwies sich als erfreulich einfach. Es musste hierfür zunächst die *struts-config.xml* erweitert und die neuen Klassen in das entsprechende Verzeichnis kopiert werden. Abschliessend musste nur noch das Menü angepasst und die Ressource-Datei um die fehlenden Einträge ergänzt werden.

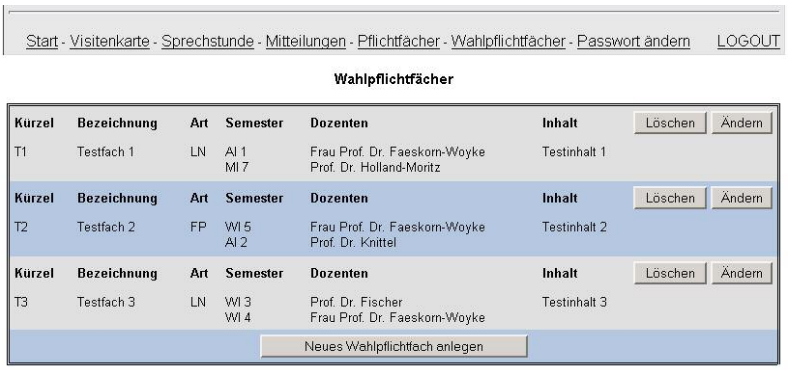

Abbildung 5.12: *Integration von HISDozent in das AMS*

Obwohl also bei der Erstellung des AMS nicht vorgesehen, führte die Verwendung von Struts zur leichten Erweiterbarkeit der Applikation. Auf der CD befindet sich die aktualisierte Version der Applikation.

5.4 Kritische Betrachtungen

Abschliessend sollen einige kritische Betrachtungen zu Struts aufgeführt werden. Auf Seite 106 werden dieselben Punkte auch bezüglich des Oracle MVC Frameworks untersucht und mit deren Umsetzung in Struts verglichen.

5.4.1 Aufwand

Struts besteht im Kern aus weniger als zehn Klassen sowie einer leicht verständlichen Tag-Bibliothek. Der Aufbau und der Ablauf innerhalb des Frameworks ist zwar zunächst ungewöhnlich, jedoch nach geringer Einarbeitungszeit zu verstehen. Erste Applikationen lassen sich so recht schnell erstellen.

Bei der intensiven Benutzung von Struts zeigt sich dann, dass dem Entwickler viele Freiräume gelassen wurden. So ist der äußere Rahmen der Applikation zwar durch die Aufteilung in Bestandteile der MVC-Architektur vorgegeben, innerhalb dieser Bestandteile bestehen jedoch recht große Spielräume, wie bei der Gestaltung des Back-End-Modells oder der Verwendung der ActionForms. Während hierdurch die Flexibilität der erstellten Applikationen gesteigert wird, bieten diese Spielräume auch ausreichend Gelegenheit, das Framework falsch umzusetzen, z.B. durch eine zu Enge Bindung zwischen Actions und ActionForms.

Um dies von vornherein zu vermeiden, ist eine intensive Auseinandersetzung mit der Dokumentation und den Ideen des Frameworks unvermeidlich, was den Einarbeitungsaufwand deutlich steigen lässt. Hier könnten strengere Vorgaben des Frameworks sicherlich hilfreich sein.

5.4.2 Trennung von Design und Implementierung

Durch die Tag-Bibliotheken ist eine Verwendung von Java-Code innerhalb der JSP nicht mehr nötig. Durch deren Zugriffsmöglichkeiten auf die ActionForms besitzen die Designer zudem ausreichend Möglichkeiten zum Gestalten der Seiten, ohne sich mit der in den Action-Klassen gekapselten Applikationslogik beschäftigen zu müssen.

Deren Implementierung in den Action-Klassen ist nicht ganz so losgelöst von den Views. Durch die Verwendung der ActionForms sowohl in den Views als auch in den Action-Klassen werden letztere relativ stark an das Front-End-Modell gebunden. Dies zeigt sich dadurch, dass eine Action zur Auswahl der auszuführenden Geschäftslogik auf die Informationen einer bestimmten ActionForm angewiesen ist. Hierdurch sind die Action-Klassen häufig nur in Verbindung mit einer bestimmten ActionForm verwendbar.

Diese enge Bindung kann teilweise durch die Verwendung von seitenübergreifenden ActionForms verhindert werden.

5.4.3 Wiederverwendbarkeit

Durch die zahlreichen Freiräume innerhalb des Frameworks können ähnliche Probleme unterschiedlich gelöst werden. Eventuell weisen hierdurch Applikationen unterschiedliche Strukturen auf, z.B. durch unterschiedliche Einbindung der Geschäftslogik in den Action-Klassen. Sollen daher auf Grundlage von Struts mehrere Applikationen erstellt werden und deren Bestandteile jeweils in hohem Maße wiederverwendbar sein, so sind hier zusätzliche Konventionen nötig, Struts alleine garantiert dies nicht.

Mit den ActionForms liegen aber auch Komponenten vor, die durch ihren Aufbau Wiederverwendbarkeit garantieren. Da die JSP zudem nicht auf die Applikationslogik zugreifen und auch das Back-End-Modell leicht wiederverwendbar gestaltet werden kann, sind sowohl Modell als auch Views insgesamt durchaus leicht wiederverwendbar. Beim technischen Controller ergibt sich dies aus der Gestaltung als ActionServlet, bei den fachlichen Controllern muss dies aus oben genannten Gründen als schwierig angesehen werden. Allerdings sind dies ohnehin die Komponenten, die bei einer Änderung oder Wiederverwendung der Applikation am ehesten zu ändern sind.

5.4.4 Nutzen

Struts ermöglicht es, dem Entwickler durch seine Benutzung eine Applikation auf Grundlage der MVC-Architektur zu erstellen. Durch seine Benutzung können somit die Vorzüge dieser Architektur, wie Flexibilität und Wiederverwendbarkeit, auch in der so erstellten Applikation angetroffen werden.

Besonders ausgeprägt ist in Struts die Zusammenarbeit zwischen den View- und Controller-Komponenten. Dies macht sich neben der Möglichkeit, mehrsprachige Applikationen erstellen und leicht Fehlermeldungen einbinden zu können, vor allem in der einfachen Einbindung komplexer Objekte in die Views bemerkbar.

Durch die zentrale Konfigurationsdatei wird zudem schon beim Entwickeln die Applikation als Einheit erkennbar, ein Umstand, der bei Web-Applikationen nicht selbstverständlich ist. Über diese Konfigurationsdatei sind auch nach Fertigstellung der Applikation noch umfangreiche Änderungen möglich, so dass eine Anpassung an unterschiedliche Umgebungen erleichtert wird.

Da auch der Quell-Code des Frameworks verfügbar ist, kann Struts zudem eigenen Bedürfnissen angepasst werden. Zwar läuft man hierdurch Gefahr, dass die Applikationen nicht mehr mit zukünftigen Struts-Versionen kompatibel sind, man hat so aber auch die Möglichkeit eigene Frameworks auf Basis von Struts zu erstellen.

Kapitel 6

Das Oracle-MVC-Framework

Zunächst wird in der Übersicht das Umfeld sowie die Ziele und die Verbreitung von Oracle MVC beschrieben.

Danach werden die einzelnen Bestandteile genauer erläutert.

Hieran anschließend wird erklärt, wie mit Oracle MVC eine Web-Applikation erstellt wird.

Anhand einiger Beispiele wird dann das Zusammenspiel der einzelnen Bestandteile erläutert.

Abgeschlossen wird das Kapitel mit einigen kritischen Betrachtungen.

6.1 Übersicht

Mit dem Oracle MVC Framework soll nun ein zweites Framework betrachtet werden, welches bei der Erstellung von Web-Applikationen auf Grundlage der MVC-Architektur verwendet werden kann. Der Schwerpunkt des Frameworks liegt hierbei in der Gestaltung des Controllers, sowie der Verwendung von unterschiedlichen Techniken innerhalb der Views und des Modells.

6.1.1 Umfeld

Oracle gilt als der weltweit größte Anbieter für Unternehmenssoftware. Zentrales Produkt hierbei ist die Oracle Datenbank. Während diese vor einigen Jahren von Oracle noch als ausreichende Basis zur Ausweitung der Unternehmenssoftware galt, wird mittlerweile hierfür zusätzlich der Oracle Application Server verwendet. Auf Basis der breiten Verwendung der Datenbanken hat Oracle im Markt eine solide Grundlage und entspricht zudem dem auf Seite 19 aufgezeigten Trend, wonach Application Server zukünftig stärker mit anderen Systemen, in diesem Fall der Datenbank, gebündelt vermarktet werden.

In diesem Zusammenhang ist sicherlich auch JHeadstart zu sehen, ein Projekt von Oracle Consulting zur Entwicklung von J2EE-Applikationen. Dies ist momentan noch nicht am Markt verfügbar, soll aber im Laufe des Jahres angeboten werden (Stand September 2002). Es läuft innerhalb des Oracle Application Servers und basiert auf drei Oracle-eigenen Frameworks, die zur Umsetzung einer MVC-Architektur kombiniert eingesetzt werden.

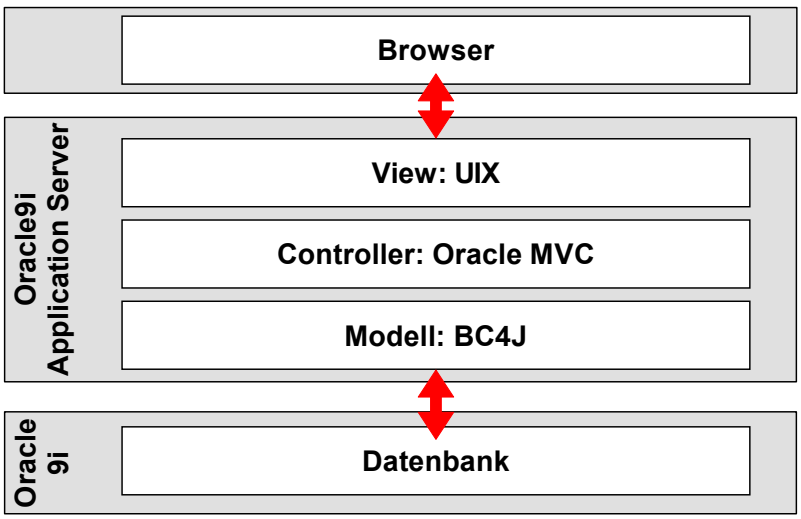

Abbildung 6.1: *Bestandteile von JHeadstart*

Für die Erstellung der Views wird innerhalb von JHeadstart User Interface XML (UIX) verwendet. UIX ist ein XML-basiertes Framework zur Beschreibung von Layout und Inhalt von HTML-Seiten. Es ist vollständig in die Oracle eigene Entwicklungsumgebung

JDeveloper integriert.

Das Modell wird mit den Business Components for Java (BC4J) erstellt. BC4J ermöglicht es Datenbanken über Objekte in beliebige Java-Applikationen einzubinden. Der Entwickler kann sich hierdurch auf das Erstellen der Applikation konzentrieren, Anforderungen an die Persistenzschicht, wie z.B. Caching oder Connection Pooling, werden von BC4J erfüllt. Auch das Erstellen und Ändern der BC4J Objekte kann vollständig innerhalb des JDevelopers durchgeführt werden.

Der Controller soll nun innerhalb von JHeadstart mit Hilfe des Oracle MVC Frameworks erstellt werden. Momentan existiert hierfür noch keine Integration in den JDeveloper, jedoch ist dies geplant. Damit könnten dann alle Bestandteile einer Web-Applikation innerhalb des JDevelopers erstellt und getestet werden.

6.1.2 Zielsetzung

Innerhalb von JHeadstart ist die Umgebung des Frameworks auf UIX für die Views sowie BC4J für das Modell festgelegt. Das Oracle MVC Framework ist jedoch sehr flexibel ausgelegt und kann in beiden Schichten mit unterschiedlichen Techniken kombiniert werden. So können in den Views neben HTML auch WML für mobile Endgeräte sowie Portlets verwendet werden. Portlets sind die Hauptbestandteile von Oracles Portal-Server. Somit können mit dem Oracle MVC Framework auch in diesem leicht Komponenten auf Basis einer MVC-Architektur integriert werden. Auf Seiten des Modells werden neben dem in JHeadstart verwendeten BC4J auch JDBC und EJB unterstützt.

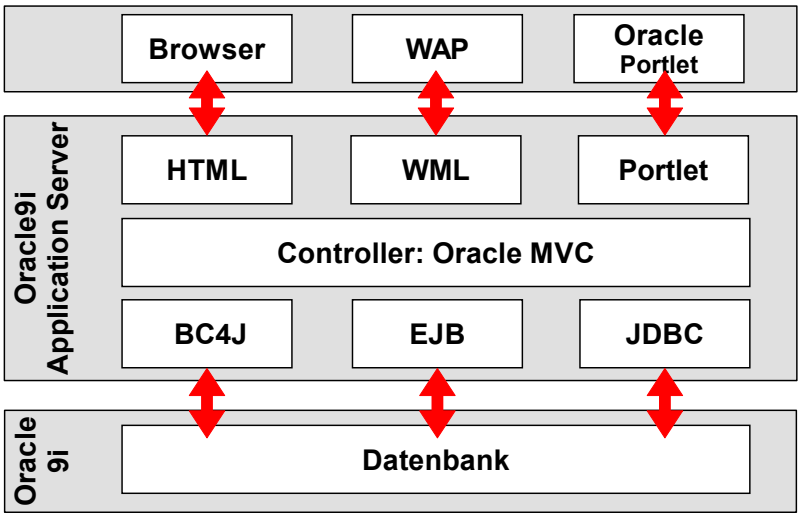

Abbildung 6.2: *Möglichkeiten des Einsatzes von Oracle MVC*

Ziel von Oracle MVC ist es nun, die Erstellung des Controllers innerhalb dieser Architektur dahingehend zu unterstützen, dass die unterschiedlichen Views und Modelle leicht

angebunden werden können. Zudem sollen die zur Erstellung des Controllers notwendigen Bestandteile leicht wiederzuverwenden sein.

Der modularisierte Aufbau des Frameworks soll es ermöglichen, beim Austausch einer View- oder Modell-Schicht in den Controller-Komponenten keine Veränderungen vornehmen zu müssen. Alle hierfür notwendigen Änderungen können deklarativ in XML-Konfigurationsdateien durchgeführt werden.

6.1.3 Verbreitung

Das Oracle MVC Framework ist frei verfügbar, die momentan aktuelle Version ist 2.0 vom September 2002. Es besitzt keine eigene Homepage, sondern ist auf den Oracle-Seiten zum Application-Server zu finden (*http://otn.oracle.com/products/ias/ias_utilities.html#mvc*). Im Oracle Technet, einer Forensammlung für Oracle-Entwickler, findet sich unter der Überschrift *Cleveland*, dem ursprünglichen Namen des Frameworks, ein eigenes Forum mit ca. 40 Beiträgen im Monat.

Hier findet sich auch die Information, dass das Framework ca. 350 mal im Monat heruntergeladen wird. Diese im Vergleich zu Struts deutlich geringere Verbreitung zeigt sich auch bei der Suche im Internet, bei der fast ausschließlich Oracle-Seiten gefunden werden.

Somit liegt neben dem Forum die Hauptinformationsquelle für das Framework in der Dokumentation [Ora02], die sich auf die Vorgängerversion (Release Candidate 2.0) bezieht.

6.2 Bestandteile von Oracle MVC

Schwerpunkt von Oracle MVC liegt in der Gestaltung des Controllers und der leichten Anbindung von unterschiedlichen Techniken in den Views und im Modell. Zum Verständnis des Frameworks sollen zunächst die von ihm zur Erstellung einer Web-Applikation verwendeten Bestandteile erläutert werden.

6.2.1 Übersicht

Das Oracle MVC Framework unterteilt den Controller einer Web-Applikation zunächst in Services, wobei ein Service einem Geschäftsvorgang entspricht. Alle für dessen Bearbeitung benötigten Schritte, wie z.B. Datenbankabfragen oder das Erstellen von Views, sind jeweils in einem Prozess enthalten. Die Reihenfolge in der die Prozesse durchlaufen werden, wird von einem speziellen Prozess, dem Router, bestimmt. Da die Reihenfolge der auszuführenden Prozesse erst zur Laufzeit bestimmt werden soll, geben die Prozesse beim Beenden ein Prozessergebnis zurück. Vorgeschrieben sind die Prozessergebnisse *success* und *failure*, es können aber auch weitere festgelegt werden. Beim Beenden eines Prozesses kann der Router nun anhand einer Transitionstabelle, in der für jeden Prozess abhängig von dessen Prozessergebnis ein Folgeprozess eingetragen ist, den nächsten auszuführenden Prozess auswählen. Es werden dann solange Prozesse ausgeführt, bis ein als Endprozess gekennzeichneter erreicht wird. Ein typischer Endprozess ist z.B. das Erstellen eines Views, da hier der Service beendet und der View zum Benutzer gesendet werden muss.

Um einen Service besser strukturieren zu können, besteht die Möglichkeit Prozessgruppen zu bilden, in denen mehrere Prozesse zusammengefasst werden. Die Reihenfolge in denen sie durchlaufen werden wird wiederum von einem Router bestimmt. Eine spezielle Prozessgruppe stellen dabei die DisplayGroups dar, die immer verwendet werden, wenn vor der Erstellung eines Views weitere Prozesse, wie z.B. Datenbankabfragen, ausgeführt werden sollen oder wenn eine Anfrage in Abhängigkeit vom Zustand der Applikation mit unterschiedlichen Views beantwortet werden soll. Eine DisplayGroup besitzt somit mindestens einen Endprozess der einen View erzeugt. Innerhalb des Services wird dann die gesamte DisplayGroup als Endprozess verwendet.

Anfragen an die Web-Applikation werden zunächst, wie in Abbildung 6.3 dargestellt, von einem ServiceController entgegengenommen. Dieser entscheidet anhand der Anfrage und einer Konfigurationsdatei, von welchem Service diese zu beantworten ist. Der Service führt nun einen als Startprozess gekennzeichneten Prozess aus. Ist dieser beendet, entscheidet der Router des Service auf Grundlage seiner Transitionstabelle und des Prozessergebnisses, welcher Prozess als nächster auszuführen ist. Zur Erstellung eines Views wird er zuletzt eine DisplayGroup aufrufen, die nun wiederum den in ihr festlegten Startprozess aufruft. Nun entscheidet der Router der DisplayGroup welche Prozesse im weiteren aufzurufen sind, bis er auf einen Endprozess stößt, der den benötigten View erzeugt. Da hiermit die DisplayGroup beendet wird, geht die Kontrolle zurück an den Service. In diesem ist wiederum die DisplayGroup als Endprozess gekennzeichnet, wes-

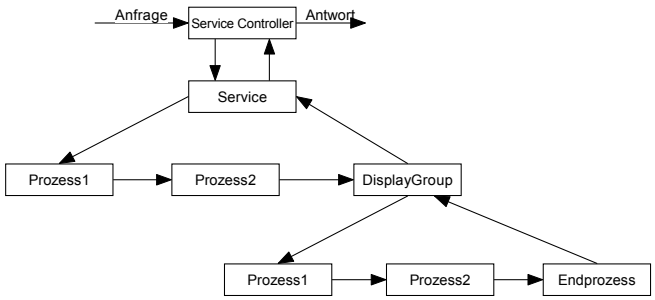

Abbildung 6.3: *Bearbeiten einer Anfrage*

halb nun die Kontrolle zurück an den ServiceController geht, der die Anfrage mit dem erstellten View beantwortet.

Nach diesem Überblick über das Framework soll nun detaillierter auf dessen Bestandteile eingegangen werden.

6.2.2 Der Controller

ServiceController

Der ServiceController stellt die Verbindung zwischen der Applikation und ihrer Umgebung dar. Er nimmt Anfragen an und startet den für sie zuständigen Service. Von diesem erhält er einen View zurück, den er an den Benutzer zurückschickt. Das Framework bietet ServiceController für HTTP, WAP sowie Oracle Portlets an. Somit können große Teile der Applikation unverändert in unterschiedlichen Umgebungen eingesetzt werden.

Service

Wie schon erwähnt, dient ein Service der Bearbeitung eines Geschäftsvorgangs innerhalb der Web-Applikation. Er kann hierfür aus Prozessen und Prozessgruppen zusammengesetzt werden. Ein Service muss nicht programmiert werden, sondern kann über eine XML-Datei vollständig deklarativ festgelegt werden. Die XML-Datei enthält neben den verwendeten Prozessen auch die Transitionstabelle für den Router. Zur Erstellung der XML-Datei kann das UML-Werkzeug Rational Rose verwendet werden. Mit einer zum Framework gehörenden Erweiterung von Rational Rose ist es möglich, den Service in einem Aktivitätsdiagramm zu modellieren und hieraus die XML-Datei generieren zu lassen. Hierdurch ermöglicht es das Framework dem Entwickler zentrale Teile der Web-Applikation über eine grafische Benutzeroberfläche zu erstellen. Der gesamte Ablauf der Applikation kann hier festgelegt, überprüft und angepasst werden.

Zum besseren Verständnis der Services soll jetzt näher auf die einzelnen Elemente der XML-Datei eingegangen werden:

- Service Tag: Hier wird der Name des Services, seine Class-Datei und der als Anfangszustand zu verwendende Prozess festgelegt. Über das validation-Attribut kann

festgelegt werden, ob die Transitionstabelle des Services beim Start überprüft werden soll.

```
<service name="Servicename"
            class="oracle.cle.process.Service"
            initial="Startprozess"
            validations="true">
</service>
```

- Process Tag: Jeder vom Service verwendete Prozess muss in einem *Process*-Tag deklariert werden. Neben seinem Namen und der zu verwendenden Class-Datei ist auch anzugeben, ob es sich um einen Endprozess handelt. Ist dies nicht der Fall, so wird der Router nach Beendigung des Prozesses auf Grundlage der Transitionstabelle den nächsten Prozess auswählen.

```
<process name="Prozessname"
      class="demo.process.TestProcess"
      end="false"
</process>
```

- ProcessGroup Tag: Innerhalb dieses Tags können logisch zusammengehörende Prozesse zu Gruppen zusammengefasst werden. Neben dem Namen der Gruppe und deren Class-Datei muss auch der Name des Startprozesses angegeben werden. Handelt es sich bei der Prozessgruppe um eine DisplayGroup, so ist sie als Endprozess zu kennzeichnen.

```
<processgroup name="Gruppenname"
              class="oracle.cle.process.DisplayGroup"
              end="true"
              initial="Startprozess">
   <process name="Prozessname1"
            class="demo.process.TestProcess1"
            end="false">
   </process>
   ...
</processgroup>
```

- InfoAlias Tag: Oft ist es nötig Parametern und Ergebnissen auf Prozessebene, Werte auf Service-Ebene zuzuordnen. Hierzu kann in einem InfoAlias Tag der Name auf Prozessebene unter *local* und der Name auf Service-Ebene unter *global* eingetragen werden:

```
<infoalias local="LokalerName" global="GlobalerName">
```

- Transition Tag: Die Transition Tags enthalten die Informationen, die der Router benötigt, um den nächsten auszuführenden Prozess bzw. die nächste auszuführende Prozessgruppe auswählen zu können. Das *src*-Attribut gibt hierbei den Prozess an, der die Kontrolle zurück an den Router übergeben hat. Das *dest*-Attribut bestimmt

den nächsten auszuführenden Prozess oder Prozess-Gruppe, falls der Wert von *condition* dem Rückgabewert des src-Prozesses entspricht:

```
<transition src="QuellProzess" condition="all"
            dest="ZielProzess"></transition>
```

Werden innerhalb des Services Prozessgruppen verwendet, so sind in diesen ebenfalls die jeweiligen Transitionen anzugeben.

- GroupTransition Tag: Wurde eine Anfrage von einer Prozessgruppe beantwortet, so wird dies vom Service abgespeichert. Kommt es nun zu einer erneuten Anfrage kann der Service anhand der GroupTransition Tags entscheiden, welcher Prozess als nächster gestartet werden soll.

```
<grouptransition srcgroup="QuellGruppe"
             srcprocess="QuellProzess"
             dest="ZielProzess"/>
```

Prozesse

Neben dem Entwickeln von eigenen Prozessen besteht die Möglichkeit auf eine Vielzahl vom Framework zur Verfügung gestellte Standardprozesse zurückzugreifen, die sich in vier Kategorien unterteilen lassen:

1. Seiten-Erstellung
   Hiermit können die Views erstellt werden. So kann z.B. mit dem Prozess *CreateJSPPage* eine JSP aufgerufen werden. Der Prozess kann vollständig in der Konfigurationsdatei des Services konfiguriert werden. Hierzu muss lediglich der Name der JSP als Argument übergeben werden. Da die Erzeugung eines Views immer ein Endprozess ist, ist zudem der entsprechende Eintrag auf *true* zu setzen:

   ```
   <process name="DozentDetailPage"
          class="oracle.clex.process.CreateJSPPage"
          end="true">
   <argument type="String"
           value="/dozentdetail.jsp"/>
   </process>
   ```

2. Validierung
   Diese Prozesse dienen der Überprüfung von Benutzereingaben. So können innerhalb des Prozesses *ValidateHTMLForm* Bedingungen für Formularfelder aufgestellt und überprüft werden. Ein Beispiel für seine Verwendung findet sich auf Seite 100.

3. Routing
   Routing-Prozesse steuern den Ablauf innerhalb eines Services oder einer Prozessgruppe. Sie reagieren dabei auf sogenannte Prozessparameter. Erhält der Service z.B. eine Anfrage aus einem HTML-Formular, so wird der Name des im Formular gedrückten Buttons global abgespeichert. Dem Router wird nun zunächst in einem Argument die Anzahl der Prozessparameter bekanntgegeben, auf die er reagieren

soll. Jedem dieser Parameter ist dann in einem *infoalias*-Tag der globale, vom Service gespeicherte, Wert zuzuordnen.

```
<process name="Router"
       class="oracle.clex.process.ProcessRouter"
       end="false">
       <argument type="Integer" value="2"/>
       ...
       <infoalias local="pp1" global="viewdozenten"/>
       <infoalias local="pp2" global="deletedozent"/>
</process>
 ...
<transition src="Router"
         condition="tc1"
         dest="DozentenListePage"/>
<transition src="Router"
         condition="tc2"
         dest="DozentenListePage"/>
...
```

Zudem muss für jeden Prozessparameter *pp* eine entsprechende Transition *tc* in der Konfigurationsdatei angegeben werden, in der der als nächstes auszuführende Prozess hinterlegt ist.

4. Email
   Ermöglichen das einfache Versenden von Emails an einen oder mehrere Empfänger.

Neben der Nutzung der Standardprozesse besteht natürlich die Möglichkeit, Prozesse selbst zu entwickeln und somit die Applikation den jeweiligen Anforderungen anzupassen. Wie eingangs erwähnt, bilden die Prozesse die Bausteine, aus denen ein Service erstellt wird. Es ist dabei bei ihrer Entwicklung besonders darauf zu achten, dass sie leicht wiederverwendbar sind. Um dies zu erreichen sollten Prozesse sehr kleinkörnig implementiert werden.

Ähnlich den Standardprozessen lassen sich auch für die selbst entwickelten Prozesse Kategorien ausmachen, welche eine Unterteilung nach deren Aufgaben vornehmen:

1. Lesende Datenzugriffe
2. Schreibende Datenzugriffe
3. Seitenanzeige
4. Ablaufsteuerung
5. Überprüfung der Geschäftslogik

Bei der Erstellung der Prozesse sollte darauf geachtet werden, dass sie nur jeweils einer dieser Kategorien entstammen, ansonsten würde ihre Wiederverwendbarkeit zu stark eingeschränkt. Aus demselben Grund ist es wichtig darauf zu achten, dass Transaktionen

immer innerhalb eines Prozesses ausgeführt werden.

InfoTable

Die Prozesse laufen innerhalb des Frameworks ohne jegliche Kenntnis von anderen Prozessen. Dennoch müssen sie die Möglichkeit haben auf Informationen zuzugreifen, die andere Prozesse erzeugt haben bzw. selber Informationen für andere Prozesse abzulegen. Ebenso müssen die Views auf entsprechende Informationen zugreifen können. Zu diesem Zweck stellt das Framework die InfoTable bereit, die auf Ebene der Services alle Ergebnisse zuvor ausgeführter Prozesse enthält. Um in der InfoTable jegliche Art von Objekt speichern zu können, ist sie als HashMap realisiert, in der Objekte vom Typ *ProcessInfo* abgespeichert werden.

Soll nun ein Prozess auf Werte in der InfoTable zugreifen, so muss er diese erst über die Methode *registerInfo* beim Service registrieren. Hierdurch kann der Service zur Laufzeit erkennen, welche Informationen vom Prozess benötigt werden. Fordert der Prozess nun über die Methode *getInfo* einen Wert aus der InfoTable, prüft der Service, ob er in dieser enthalten ist. Ist dies nicht der Fall, sucht der Service in der aktuellen vom Client erhaltenen Anfrage nach ihm und speichert ihn in der InfoTable ab.

6.2.3 Das Modell

Um innerhalb der Prozesse auf das Modell zugreifen zu können, wird dieses durch spezielle Interfaces, die sogenannten Ressourcen, repräsentiert.

Ressourcen und RessourceHandler

Bei der Verwendung von Datenbanken wird eine Ressource in der Regel einer Tabelle entsprechen. Sie besitzt dann lediglich Methoden zum Lesen und Schreiben der jeweiligen Attribute. In den Ressourcen ist also keine Information über die jeweils verwendete Persistenztechnik, wie z.B. JDBC oder BC4J, enthalten. Der Entwickler kann sich somit innerhalb der Prozesse auf das Erstellen der Applikationslogik konzentrieren. Aufgaben wie Transaktionen und Caching werden hierbei von der jeweiligen Persistenz-Schicht übernommen. Durch die Benutzung der Interfaces ist man zudem nicht gezwungen sich auf eine bestimmte Persistenz-Schicht festzulegen, vielmehr kann diese auch später noch leicht ausgetauscht werden.

Um nun einzelne Ressource-Objekte erhalten zu können, ist die Benutzung sogenannter Handler-Interfaces notwendig. Sie dienen dazu einzelne oder Gruppen von Ressource-Objekten zu erhalten oder abzuspeichern. Auch hier dient die Benutzung von Interfaces wieder der Steigerung der Flexibilität der Applikation. Eine Benutzung von Klassen mit implementierten Persistenzmechanismen an dieser Stelle würde die Applikation an die verwendete Persistenztechnik binden und ein späteres Austauschen erschweren.

Implementierung der Persistenzschichten

Für die oben beschriebenen Interfaces müssen natürlich auch entsprechende Implementierungen erstellt werden. Hierzu müssen für die zu verwendende Persistenztechnik Value-Objekte erstellt werden, welche das jeweilige Interface implementieren. Abbildung 6.4 soll den Zusammenhang zwischen Ressourcen, Ressource-Handlern und Persistenzschicht verdeutlichen.

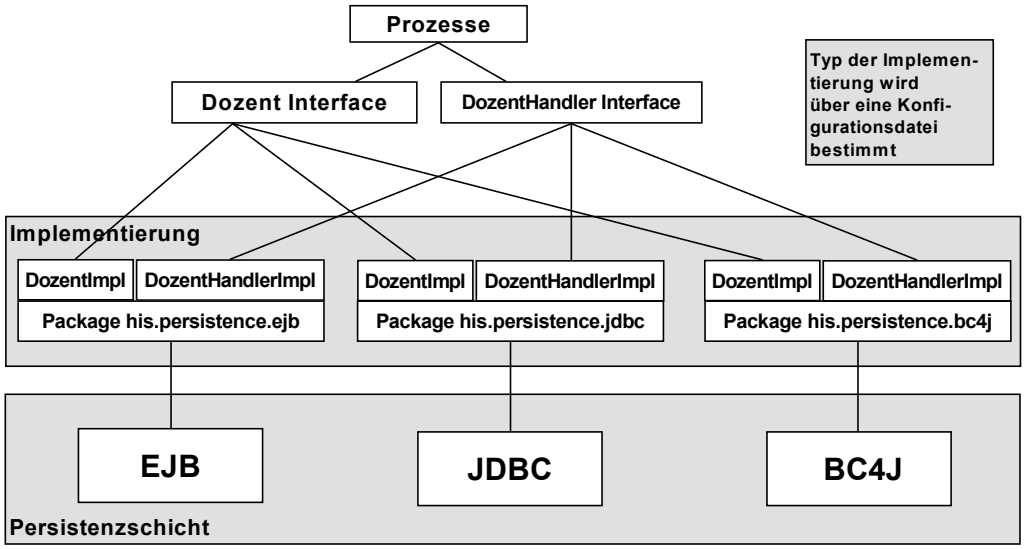

Abbildung 6.4: *Zusammenhang zwischen Ressourcen, Handlern und Persistenzschicht*

Bei der Erstellung der Applikation wird man zunächst die Ressourcen und Ressource-Handler erstellen. Diese sind ausreichend um die benötigten Prozesse zu entwickeln. Die Verbindung zur jeweiligen Persistenzschicht wird dann in den Implementierungen vorgenommen, die innerhalb des Packages *servicename.persistence.persistenztechnik* zu erstellen sind und die in den Ressourcen und RessourceHandlern definierten Interfaces umsetzen. Dies sind dann auch die einzigen Klassen, die bei der Verwendung einer anderen Persistenzschicht zu ändern sind, die restliche Applikation bleibt hiervon unberührt.

6.2.4 Die Views

Natürlich ist die Erstellung der Views abhängig von dem für den Service verwendeten ServiceController. So sind für WML-Seiten andere Views zu erstellen als für HTML-Seiten. Um den Fokus weiterhin auf Web-Applikationen zu belassen, soll im Folgenden jedoch nur auf die Erstellung von HTML-Views eingegangen werden.

Bei der Beschreibung der Prozesse wurde gezeigt, dass diese alle benötigten und erzeugten Informationen in der InfoTable ablegen. Möchte man nun Views für eine Web-Applikation erzeugen, so benötigt man eine Möglichkeit, auf die InfoTable lesend und schreibend zugreifen zu können. Innerhalb von J2EE bieten hierfür JSPs die optimale

Voraussetzung. Und so stellt auch das Framework eine entsprechende Tag-Bibliothek zur Verfügung. So kann über die *set*- und *getInfoTableObject*-Tags innerhalb der Views auf Objekte in der InfoTable zugegriffen werden. Zudem stehen für jedes in HTML-Formularen vorkommende Element, wie z.B. Auswahllisten oder Checkboxen, entsprechende Tags zur Verfügung, mit denen diese Elemente mit Objekten oder Objektattributen in der InfoTable in Verbindung gebracht werden können.

6.3 Erstellen einer Web-Applikation mit Oracle MVC

Für die Beschreibung des Entwicklungsprozesses bietet sich ein Teil der auf Seite 63 beschriebenen Web-Applikation HISDekanat an. Zur besseren Übersicht sollen lediglich Vor- und Nachname der Dozenten angezeigt, geändert und gelöscht werden können.

6.3.1 Konfiguration von Tomcat

Die Applikation soll auch wieder innerhalb des Tomcat-Servers erstellt werden. Hierzu ist zunächst das Verzeichnis */webapps/hisdekanatoramc* anzulegen und die Dateien des Frameworks in das Verzeichnis */webapps/hisdekanatoramc/WEB-INF/lib* zu kopieren. In der Datei */WEB-INF/web.xml* sind folgende Einträge vorzunehmen:

```
<servlet>
  <servlet-name>hisdekanat</servlet-name>
  <servlet-class>
    oracle.clex.process.controller.HttpServletController
  </servlet-class>
  <init-param>
    <param-name>cle-service-descriptor</param-name>
    <param-value>hisdekanat.xml</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>hisdekanat</servlet-name>
  <url-pattern>/dozenten</url-pattern>
</servlet-mapping>
```

Hierdurch werden sämtliche Anfragen an die Adresse *./hisdekanatoramvc/dozenten* an den ServiceController weitergeleitet. In der Datei *hisdekanat.xml* wird dann der Service konfiguriert.

6.3.2 Erstellen des Controllers Teil 1

Ähnlich zu Struts soll auch hier zunächst die Konfigurationsdatei für die Applikation erstellt werden. Oracle bietet hierfür eine Erweiterung für das UML-Tool Rational Rose an. Mit dieser kann die Konfigurationsdatei direkt aus einem in Rose erstellten Aktivitätsdiagramms generiert werden.

Das Aktivitätsdiagramm stellt hierbei den gesamten Service dar. Prozesse und Prozessgruppen werden als Aktivitäten in das Diagramm eingetragen. Zusätzliche Konfigurationsparameter können über Notizen mit ihnen verknüpft werden. Transitionen werden durch Pfeile symbolisiert.

Für HISDekanat ist zunächst ein HISDekanatRouter in das Diagramm einzutragen. Da dieser alle Anfragen an den Service entgegen nehmen soll, ist er durch einen Startzustand

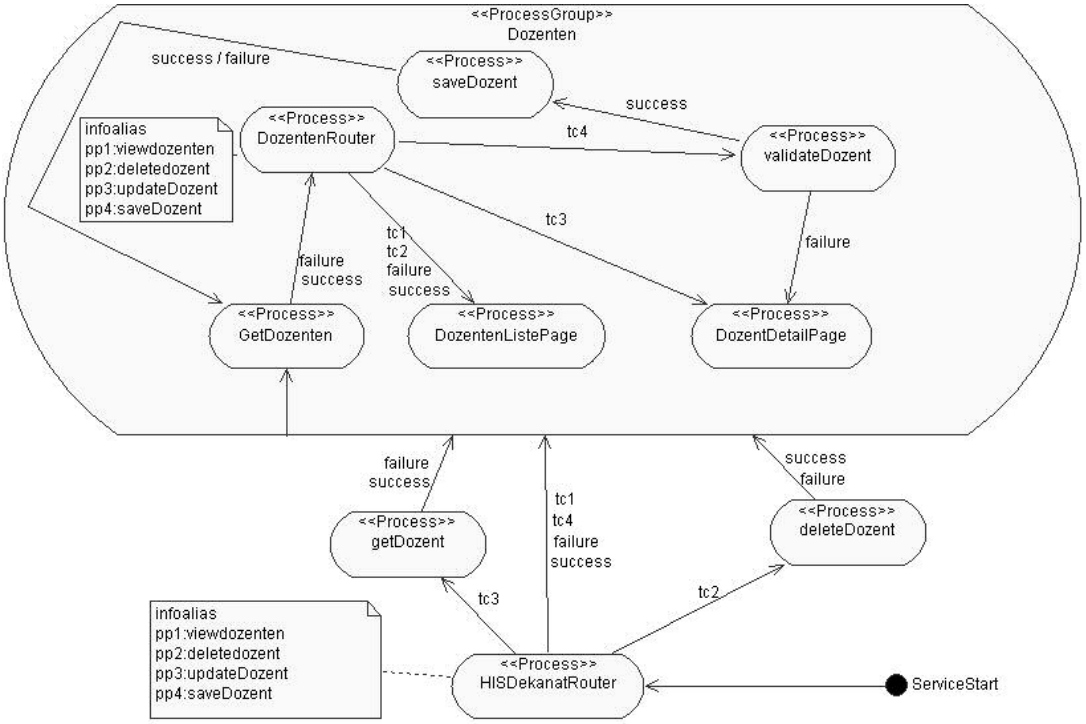

Abbildung 6.6: *Aktivitätsdiagramm für den Service Dozenten verwalten*

zu kennzeichnen. Mit einem Doppelklick gelingt man auf das in Abbildung 6.7 dargestellte Eigenschaftsfenster des Routers, in dem zunächst Name und Typ festgelegt werden. Im Fenster *Documentation* wird dann als erstes die Klasse des Prozesses angegeben. Der zweite Wert gibt an, ob es sich um einen Endzustand handelt oder nicht. Anschließend wird die Anzahl der Prozessparameter angegeben, danach ob der Router nach dem Bearbeiten einer Anfrage Informationen in der InfoTable löschen soll und schließlich der Standardzustand des Routers. Die vier Prozessparameter auf die HISDekanatRouter reagieren soll, sind dann über eine Notiz mit dem Router zu verknüpfen.

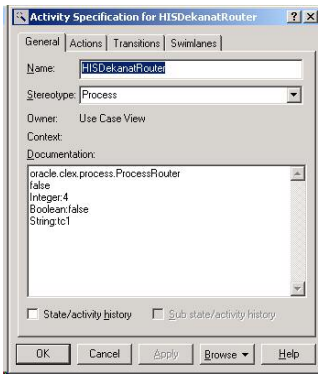

Abbildung 6.7: *Eigenschaftenfenster für HISDekanatRouter*

Auf Service-Ebene werden noch zwei zusätzliche Prozesse benötigt. *GetDozent* zum Auslesen eines Dozenten aus der Datenbank, sowie *deleteDozent* zum Löschen eines Dozenten.

Als nächstes wird eine Prozessgruppe *Dozenten* angelegt, welche alle die Verwaltung der Dozenten betreffenden Views, sowie die hierfür benötigten Prozesse enthält. Sie ist somit als DisplayGroup anzulegen. Um den Ablauf innerhalb der DisplayGroup steuern zu können, wird wieder ein Router benötigt, dessen Konfiguration analog zu der oben beschriebenen verläuft.

Nun können die restlichen in der Prozessgruppe benötigten Prozesse in das Diagramm eingetragen werden. In der Prozessgruppe *Dozenten* wird als erstes der Prozess *GetDozenten* erstellt, der alle in der Datenbank enthalten Dozenten auslesen soll. Er ist über eine Transition mit der Prozessgruppe verbunden, wodurch er als Standardprozess gekennzeichnet wird. Für das Speichern von Dozenten dient der Prozess *saveDozent*, da die Daten jedoch vor dem Speichern geprüft werden sollen, wird diesem der Prozess *validateDozent* vorgeschaltet. Zur Erstellung der Views dienen schließlich die Prozesse *DozentListePage* sowie *DozentDetailPage*. Da sie auf JSP zugreifen sollen, wird für sie der Standardprozess *CreateJSPPage* verwendet, der vollständig in den Eigenschaftsfenstern konfiguriert werden kann:

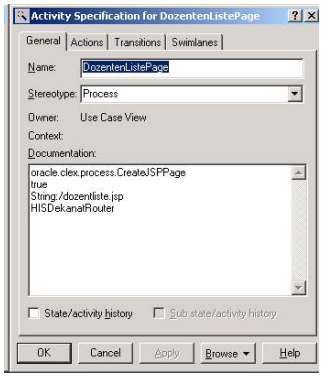

Abbildung 6.8: *Eigenschaftenfenster für DozentListePage*

Nachdem alle Prozesse in das Diagramm eingetragen wurden, können nun die Transitionen zwischen ihnen erstellt werden. Bei den meisten Prozessen ist es ausreichend die Transitionen *success* und *failure* festzulegen. Bei den Routern muss zusätzlich für jeden in den Notizen festgelegten Zustand *ppx* eine Transition *tcx* in das Diagramm eingetragen werden. So soll der HISDekanatRouter eine Anfrage bei den Zuständen *viewDozenten* und *saveDozent* direkt an die Prozessgruppe *Dozenten* weiterleiten, während z.B. beim Zustand *deleteDozent* noch der gleichnamige Prozess zwischengeschaltet wird. Alle in die Prozessgruppe gehenden Transitionen müssen nun auch von dessen Router weitergeleitet werden. So führen *deleteDozent* und *viewDozenten* auf den Prozess *DozentListePage* und somit zur Anzeige der Dozenten. *updateDozent* soll das Ändern eines Dozenten ermöglichen und führt somit auf eine Detailseite. Die hier gemachten Änderungen werden beim Abspeichern des Dozenten zunächst im Prozess *validateDozent* geprüft. Kommt es

zu Fehlern wird erneut die Detailseite angezeigt, ansonsten wird der Prozess *saveDozent* aufgerufen.

Mit dem Erstellen der Transitionen ist das Aktivitätsdiagramm vollständig und es kann über die von Oracle zur Verfügung gestellte Erweiterung die Konfigurationsdatei erstellt werden, welche unter */WEB-INF/classes/hisdekanat.xml* abzuspeichern ist. Sie ist auch im Anhang auf Seite VI zu finden.

6.3.3 Erstellen des Modells

Bevor die noch fehlenden Prozesse implementiert werden können, muss erst das Modell erstellt werden. Um die Web-Applikation in Tomcat ablaufen zu lassen, wurde JDBC als Technik gewählt. Hierfür ist zunächst in der Datei */WEB-INF/classes/deployment.properties* der Standardwert zu ändern:

```
default=hisdekanat_jdbc
```

Die entsprechenden Angaben für die Datenbankverbindung sind dann in der Datei */WEB-INF/classes/cle-provider.xml* einzutragen:

```
<cle-providers>
  <provider name="hisdekanat_jdbc"
    class="oracle.clex.persistence.jdbc.JDBCConnectionProvider">
   <property name="user" value="his"/>
   <property name="password" value="hisuser"/>
   <property name="encryptiontype" value="none"/>
   <property name="drivertype"
          value="oracle.jdbc.driver.OracleDriver"/>
   <property name="connectionstring"
   value="jdbc:oracle:thin:his/hisuser@localhost:1521:dbfh"/>
   <property name="host" value="localhost"/>
   <property name="sid" value="dbfh"/>
   <property name="port" value="1521"/>
   <property name="persistencebase"
          value="hisdekanat.persistence.jdbc"/>
   <property name="failover" value=""/>
  </provider>
</cle-providers>
```

Für die Applikation wird lediglich die Ressource *Dozent* benötigt. Hierfür ist zunächst das gleichnamige Interface zu erstellen, in dem alle benötigten Methoden enthalten sind:

```
package hisdekanat.resource;
public interface Dozent
{
  public String getNachname();
  public void setNachname(String name);
  public String getVorname();
  public void setVorname(String s);
  public String getKuerzel();
```

```
  public void setKuerzel(String s);
}
```

Dieses Interface ist nun in der Klasse *hisdekanat.persistence.DozentValueObject* einzubeziehen und die Methoden auszuprogrammieren. Im nächsten Schritt werden nun die Handler-Klassen erstellt. Zunächst wird hierfür wieder ein Interface benötigt, welches alle notwendigen Methoden definiert:

```
package hisdekanat.resource;

import java.util.*;
import oracle.cle.persistence.*;
import oracle.cle.resource.*;

public interface DozentHandler extends HandlerDefinition
{
  public void updateDozent(String kuerzel,
                    String vorname,
                    String nachname,
                    Object sessionid)
                    throws Exception;
}
```

Natürlich ist nun auch wieder die entsprechende Umsetzung des Interfaces für JDBC zu erstellen. Das Interface *HandlerDefinition* definiert hierbei einige Standardmethoden, die zusätzlich zu erstellen sind. So dient die Methode *deleteObject* zum Löschen des jeweiligen Objekts, welches anhand des übergebenen Schlüssels eindeutig festgelegt ist:

```
public class DozentHandlerImpl extends Handler
                        implements DozentHandler
{
...
public void deleteObject(Object key, Object sessionId)
     throws Exception
{
  final String deleteString =
             "delete from dozent where dozentkuerzel=?";
  try
  {
    Connection conn = (Connection)getConnection(sessionId);
    PreparedStatement pstmt = conn.prepareStatement(deleteString);
    pstmt.setString(1,(String)key);
    pstmt.executeUpdate();
    releaseConnection(sessionId);
  }
  catch(Exception e)
  {
    throw e;
  }
  finally
```

```
  {
    releaseConnection(sessionId);
  }
}
...
}
```

Zusätzlich sind die Methoden *findObject* für das Suchen eines Dozenten, *getAllObjects* für einen Vector mit allen Dozenten, sowie *updateDozent* für das Aktualisieren eines Dozenten zu erstellen. Bei allen Methoden wird das Connection-Management vom Framework übernommen, von dem über die Session-ID Datenbankverbindungen erhalten werden können.

6.3.4 Erstellen des Controllers Teil 2

Es können nun die noch fehlenden Prozesse erstellt werden. Zunächt der Process *GetDozenten*, der alle Dozenten in einem Vector abspeichern und in die InfoTable ablegen soll. Da der Prozess hierfür auf die Persistenzschicht zugreifen muss, empfiehlt es sich ihn von *PersistingProzess* abzuleiten. Dass von ihm erzeugte Prozessergebnis ist dann in der Methode *registerInfo* beim aufrufenden Service zu registrieren.

```
package hisdekanat.process;
...
public class GetDozenten extends PersistingProcess
{
...
  public void registerInfo()
  {
     ProcessResult dozenten =
       new ProcessResult("Dozenten",
                    "java.util.Vector",
                    "Vector mit allen Dozenten",
                    this, null);
     registerResult(dozenten);
  }
...
  public void persist() throws Exception
  {
    Vector dozenten = new Vector();
      DozentHandler dozentHandler =
      (DozentHandler)HandlerFactory.getHandler(Dozent.class);
    dozenten = dozentHandler.getAllObjects(getSessionId());
    setResult("Dozenten", dozenten);
  }
}
```

Die eigentliche Aufgabe des Prozesses ist abschließend in der Methode *persist* zu erstellen. Durch die Verwendung der Interfaces im Modell kann die Methode ohne Verände-

rungen mit unterschiedlichen Persistenztechniken eingesetzt werden.

Dasselbe gilt auch für den Prozess *GetDozent*, der anhand des Dozentenkürzels einen Dozenten aus der Datenbank ausliest und in der InfoTable speichert. Da er hierfür nicht nur schreibend, sondern auch lesend auf die InfoTable zugreifen muss, ist in der Methode *registerInfo* auch der zu lesende Parameter bekanntzumachen:

```
...
  protected void registerInfo()
  {
     registerStringParameter("dozentkuerzel");
     registerStringResult("dozentvorname");
     registerStringResult("dozentnachname");
  }
...
  public void persist() throws Exception
  {
    String dozentKuerzel = getStringInfo("dozentkuerzel");
    Dozent dozent=null;
    Object sessionId = getSessionId();
    DozentHandler dozentHandler=
       (DozentHandler)HandlerFactory.getHandler(Dozent.class);
    dozent =
       (Dozent)dozentHandler.findObject(dozentKuerzel,sessionId);
    if(dozent!=null)
    {
      setResult("dozentnachname",dozent.getNachname());
      setResult("dozentvorname",dozent.getVorname());
    }
  }
...
```

Im Vergleich zu Struts fällt hierbei besonders auf, dass Oracle MVC kein Front-End-Modell verwendet. Alle von den Views benötigten Informationen, die bei Struts ja in den ActionForms gespeichert wurden, werden in der InfoTable abgelegt. Als Folge hiervon können die Prozesse mit unterschiedlichen Techniken in den Views verwendet werden und sind nicht an eine spezielle Technik gebunden. Ein Nachteil ergibt sich in der notwendigen Registrierung der jeweils verwendeten Parameter, deren Aufwand bei umfangreichen Views nicht zu unterschätzen ist. Da hierbei die Parameter, z.B. bei den Methoden *registerStringResult* und *setResult*, über Strings festgelegt werden, können Schreibfehler nicht zur Compile-Zeit erkannt werden, sondern treten erst zur Laufzeit auf.

Da die Prozesse *saveDozent* und *deleteDozent* analog zu den oben beschriebenen erstellt werden, soll abschließend noch auf den Prozess *ValidateDozent* eingegangen werden. Dieser soll vor dem Speichern eines Dozenten prüfen, ob Vor- und Nachname angegeben worden sind. Hierfür empfiehlt es sich den Standardprozess *ValidateHTMLForm* des Frameworks abzuleiten, da dieser spezielle Methoden zum Überprüfen von Eingabedaten anbietet:

```
...
public class ValidateDozent extends ValidateHTMLForm{
...
 protected void registerInfo()
 {
  registerStringParameter("dozentvorname");
  registerStringParameter("dozentnachname");
 }
...
 public void registerValidation()
 {
  registerMandatoryField("dozentvorname");
  registerMandatoryField("dozentnachname");
 }
...
}
```

Zunächst sind in der Methode *registerInfo* wieder die aus der InfoTable benötigten Parameter zu registrieren. In der Methode *registerValidation* können nun für die einzelnen Felder Bedingungen aufgestellt werden. So erzwingt z.B. *registerMandatoryField* das Vorhandensein des entsprechenden Feldes. Für weitergehende Überprüfungen bietet die Klasse *ValidateHTMLForm* unter anderem Methoden zum Prüfen von Wertebereichen oder Datumsformaten an.

Auch hier zeigt sich wieder die stärkere Modularisierung des Oracle MVC Frameworks gegenüber Struts. Bei diesem wurden die Validierungen ja innerhalb der ActionForms und zum Teil auch den Actions durchgeführt, während hierfür im Oracle MVC Frameworks eigene Prozesse erstellt werden.

6.3.5 Erstellen der Views

Abschließend sind nun noch die Views zu erstellen. Oracle MVC bietet hierfür einige Tags an, die es ermöglichen in der InfoTable enthaltene Daten leicht in den Views auszugeben. So kann die Liste der Dozenten, die von der Methode *GetDozenten* in die InfoTable gespeichert wurde, in der Seite *dozentenliste.jsp* über den *tableTag* ausgegeben werden:

```
<tabletags:showTable infoName="Dozenten"
                attributes="kuerzel,nachname,vorname"
                columnNames="Loeschen,Vorname,Nachname"
                inputNames="dozentkuerzel,vorname,nachname"
                displayParameters="Radio,text,text"
                formatParameters="LEFT,LEFT,LEFT"
                columnWidths="40,300,300"
                clear="false"
                tableParameters="border=\"1\""/>
```

Neben den auszugebenden Attributen können hier auch die zu erstellenden Formularelemente und Formatierungen angegeben werden. Das Oracle MVC Framework bietet

überwiegend Tags zur Erzeugung von Formularelementen an, so dass eventuell bei der Erstellung von Präsentationslogik, im Unterschied zu Struts, auf Java-Code zurückgegriffen werden muss.

Anhand der Seite *dozentdetails.jsp*, in der Vor- und Nachname von Dozenten geändert werden sollen, soll nochmal das Zusammenspiel zwischen Views und Controller erläutert werden:

```
<jsp:useBean id="infobean"
          class="oracle.clex.process.jsp.GetInfoBean"/>

<FORM method="POST"
     action="<%= infobean.getStringInfo("controller") %>">
<taglib:text type="hidden"
          name="dozentkuerzel"
          attribute="dozentkuerzel"/>
Vorname:<taglib:text type="text"
                name="dozentvorname"
                attribute="dozentvorname"/>
Nachname:<taglib:text type="text"
                name="dozentnachname"
                attribute="dozentnachname"/>
<input type="submit"
      name="saveDozent"
      value="Aenderungen speichern"/>
</FORM>
```

Über die *InfoBean* wird zunächst die Zieladresse für das Formular ausgelesen. Danach werden über die entsprechenden Tags die Formularfelder für das Kürzel, sowie den Vor- und Nachnamen in die Seite geschreiben. Danach wird der Submit-Button erstellt, der hierbei verwendete Name *saveDozent* wird dann beim Absenden des Formulars vom HIS-DekanatRouter entgegengenommen und die Applikation anhand des Zustandsautomaten fortgesetzt.

6.3.6 Absichern der Applikation

Die Entwicklung der Applikation endet mit dem Absichern des Services. Hierfür ist es möglich jedem Prozess und jeder Prozessgruppe innerhalb des Services, eine weitere Prozessgruppe vorzuschalten in der die Berechtigungen des jeweiligen Benutzers überprüft werden. Die zu verwendende Prozessgruppe ist hierbei über das *include*-Attribute festzulegen:

```
...
 <processgroup name="Dozenten"
            class="oracle.cle.process.DisplayGroup"
            end="true"
            initial="GetDozenten"
```

```
            included="pre-security.xml">
...
```

Bevor nun die Prozessgruppe *Dozenten* gestartet wird, wird erst die in der Datei *pre-security.xml* festgelegte Prozessgruppe ausgeführt. Kann hierbei die Berechtigung des Benutzers für den Service festgestellt werden, geht es wie gewohnt in diesem weiter. Anderenfalls können in der *pre-security.xml* Fehlerseiten festgelegt werden die dem Benutzer angezeigt werden sollen.

Im Unterschied zu Struts ist somit ein Sicherheitskonzept in das Framework integriert worden. Mit diesem lassen sich sämtliche von außen erreichbare Bestandteile der Applikation sehr flexibel schützen. Durch die Verwendung von Konfigurationsdateien können diese Schutzmechanismen zudem an zentraler Stelle angepasst werden.

6.4 Das Zusammenspiel der Bestandteile

Die Zusammenhänge zwischen den einzelnen Bestandteilen der Applikation sollen nun am Beispiel verschiedener Anfragen nochmals verdeutlicht werden.

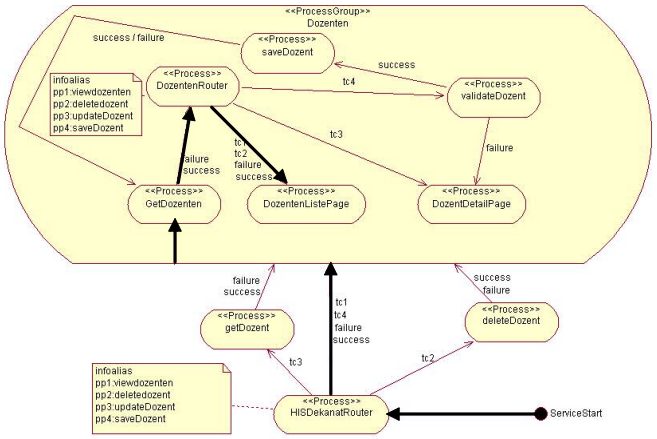

Abbildung 6.9: *Bearbeiten einer Anfrage*

Die Anfrage an die Adresse *../hisdekantoramvc/dozenten* wird, wie in Abbildung 6.9 gezeigt, aufgrund der Einstellungen in der Datei *web.xml* an den HISDekanatRouter weitergeleitet. Da in der Anfrage kein Parameter enthalten ist, startet der Router den festgelegten Standardprozess, in diesem Fall die Prozessgruppe *Dozenten*. In dieser wird nun wiederum deren Standardprozess *GetDozenten* ausgeführt, der alle Dozenten aus der Datenbank ausliest und in der InfoTable speichert. Nach dessen Beenden wird vom *DozentRouter* der Endprozess *DozentListePage* aufgerufen, der alle in der InfoTable abgespeicherten Dozenten als Liste anzeigt.

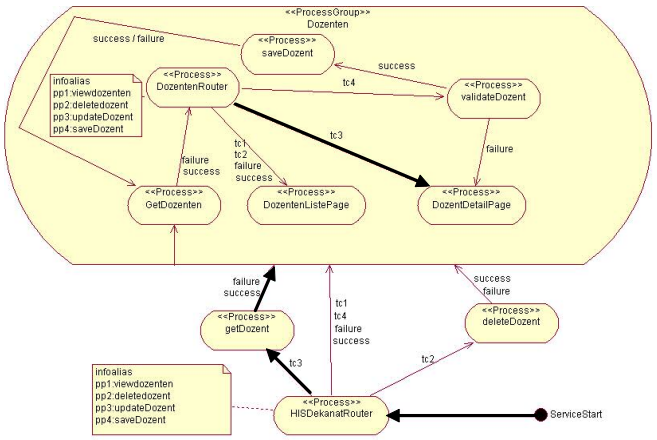

Abbildung 6.10: *Bearbeiten der Anfrage updateDozent*

In dieser Liste kann der Benutzer einen Dozenten auswählen und sich über den But-

ton *Dozent ändern* Details anzeigen lassen. Der Name des Buttons ist *updateDozent* und wird bei einer erneuten Anfrage wiederum vom HISDekanatRouter entgegengenommen (s. Abbildung 6.10). Da für den Parameter *updateDozent* eine Transition hinterlegt ist, wählt der Router den Prozesse *GetDozent* aus, der anhand des in der Anfrage enthaltenen Dozentenkürzels den entsprechenden Dozenten aus der Datenbank ausliest und dessen Vor- und Nachname in die InfoTable schreibt. Danach wird erneut die Prozessgruppe *Dozenten* aufgerufen, deren Router nun, wieder mittels des Parameters *updateDozent*, den Prozess *DozentDetailPage* aufruft, der einen View anzeigt in dem die Daten geändert werden können.

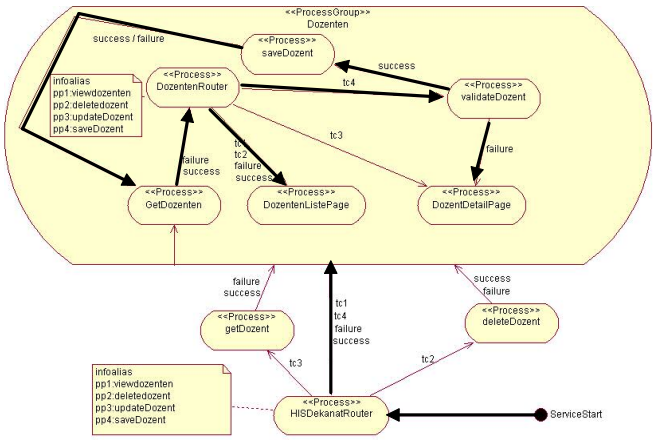

Abbildung 6.11: *Bearbeiten der Anfrage saveDozent*

Klickt der Benutzer in diesem View auf den Button *Änderungen speichern*, so werden seine Eingaben wieder an den *HISDekanatRouter* gesendet (s. Abbildung 6.11). Anhand des Parameters *saveDozent* wählt dieser wieder die Prozessgruppe *Dozenten* aus. In dieser werden die Daten zunächst durch den Prozess *validateDozent* überprüft. Entstehen hierbei Fehler, so wird erneut die Detailseite angezeigt. Anderenfalls wird der Prozess *saveDozent* ausgeführt, der die Änderungen in der Datenbank abspeichert. Um abschließend die aktualiserte Liste der Dozenten anzuzeigen, geht es von diesem über den Prozess *GetDozenten* und den Router wieder zur *DozentListePage*.

6.5 Kritische Betrachtungen

Analog zu der Beschreibung von Struts sollen auch hier verschiedene Kriterien bei der Benutzung von Oracle MVC betrachtet werden. Zudem soll falls möglich, Vergleiche mit Struts vorgenommen werden.

6.5.1 Aufwand

Zur Anwendung von Oracle MVC sind stellenweise tiefergehende Kenntnisse der J2EE-Platform notwendig als bei der Verwendung von Struts. Grund hierfür ist die explizite Verwendung von zusätzlichen J2EE-Bestandteilen, wie zum Beispiel EJB oder BC4J.

Dies ist sicher auch einer der Gründe, warum das Framework innerhalb von JHeadstart wahrscheinlich im Verbund mit Beratungsleistungen von Oracle Consulting angeboten wird. Es mag zunächst widersprüchlich erscheinen, zur Erstellung von Web-Applikationen ein Framework zu verwenden, für dessen Benutzung eine Beratung sinnvoll erscheint. Es ist jedoch zu erwarten, dass der hierdurch erlangte Nutzen den Aufwand rechtfertigen wird. Zumal auch Open-Source-Frameworks wie Struts nicht ohne Einarbeitungsaufwand sinnvoll einzusetzen sind.

6.5.2 Trennung von Design und Implementierung

Design und Implementierung können in Oracle MVC als fast vollständig getrennt angesehen werden. Es kommt hierbei kaum zu Überschneidungen. Dies ist auch notwendig, da das Framework in den Views unterschiedliche Techniken unterstützt. Somit ist es im Unterschied zu Struts möglich, neben HTML z.B. auch WAP für mobile Anwendungen zu unterstützen. Es müssen hierfür lediglich die entsprechenden Views erstellt und die Prozesse, von denen sie erzeugt werden, in der Konfigurationsdatei festgelegt werden. Die restlichen Bestandteile der Applikation können unverändert weiterverwendet werden.

6.5.3 Wiederverwendbarkeit

Das Ziel der Wiederverwendbarkeit ist bei Oracle MVC in allen Komponenten deutlich zu erkennen. Durch den Einsatz zahlreicher Schnittstellen wird es somit ermöglicht, sowohl Controller- als auch Modell-Komponenten leicht wiederverwenden zu können. Gerade bei den Prozessen, als kleinste Controller-Komponenten durchaus den Action-Klassen des Struts Frameworks vergleichbar, fällt dies gegenüber Struts deutlich auf.

Die starke Komponentenbildung, die durch Oracle MVC geradezu erzwungen wird, ermöglicht es zudem weitere Applikationen nahezu im Baukastensystem zusammenzusetzen. Im Unterschied zu Struts haben hiermit erstellte Applikationen zudem eine zu großen Teilen identische Struktur, da der Rahmen, in dem sich der Entwickler bei der Erstellung der Applikation bewegen kann, deutlich enger gesteckt ist. Hierdurch besitzen die erstellten Applikationen eventuell eine deutlich höhere Uniformität als bei Struts, können also

insbesondere leichter verglichen und verstanden werden.

6.5.4 Nutzen

Beim Einsatz des Frameworks werden häufig wichtige Bestandteile des Modells schon vorliegen. Aus vorangehenden Projekten existieren eventuell auch schon einige Prozesse bzw. es kann auf Standardprozesse zurückgegriffen werden. Ein Großteil der Applikation kann dann über die grafische Benutzeroberfläche erstellt werden, ohne Programm-Code schreiben zu müssen.

Zu implementierende Komponenten können aufgrund der Struktur des Frameworks leicht identifiziert werden. Die in ihnen zu erstellenden Programmteile sind zudem durch das Framework streng umrissen. Sicherheitsabfragen können an zentraler Stelle leicht in die Applikation eingebunden werden.

Oracle MVC nutzt hierbei die Komponentenbildung deutlich stärker zur Erstellung der Applikation als Struts. Zudem ist es bei der Einbindung von grafischen Benutzeroberflächen schon einen Schritt weiter, wobei beim Erstellen mehrerer Applikationen innerhalb desselben Kontextes die Entwicklung vollständig in diesen durchgeführt werden kann.

Kapitel 7

Fazit

Das Thema *Frameworks bei der Entwicklung von Web-Applikationen am Beispiel von Jakarta Struts und Oracle MVC Framework* erforderte zunächst eine Eingrenzung des Begriffs Web-Applikation. Die hieraus resultierenden Anforderungen ließen bereits die Komplexität erahnen, mit der bei ihrer Entwicklung zu rechnen ist.

Bei der Untersuchung der bei der Entwicklung einsetzbaren Techniken zeigte sich, dass clientseitig immer noch HTML dominiert. Da sich hiermit jedoch keine Applikationslogik erstellen lässt, liegt der Schwerpunkt der Entwicklung von Web-Applikationen weiterhin bei den serverseitigen Techniken. Eine dieser Techniken ist J2EE, welches neben den Web- auch die Application-Server in die Entwicklung mit einbezieht. Hierdurch ergeben sich eine Vielzahl von Möglichkeiten zur Gestaltung von Web-Applikationen. Um aus diesen sinnvolle und der jeweiligen Problemstellung angemessene Lösungen entwickeln zu können, sind erprobte Vorgehensweisen notwendig, lediglich der Einsatz von J2EE garantiert dies nicht.

Eine dieser Vorgehensweisen besteht in der Verwendung von Frameworks. Sie geben einen Rahmen für die Entwicklung von Applikationen vor, in dem diese zur sinnvollen Umsetzung der verwendeten Techniken durchgeführt werden kann. Diese Rahmen sind hierbei weit genug, um die Applikation spezifischen Anforderungen anpassen zu können, aber auch ausreichend eng, um die vom Framework gesetzten Ziele und Vorteile zu erreichen. In dieser Arbeit wurde eine Vielzahl von Frameworks auf Basis von J2EE vorgestellt. In ihrer Gesamtheit zeigen sie, dass für eine große Anzahl von Web-Applikationen auf bestehende Lösungen zurückgegriffen werden kann. Gleichzeitig wird aber auch nochmals die Notwendigkeit von Frameworks deutlich.

Mit Struts und Oracle MVC wurden dann zwei dieser Frameworks detaillierter betrachtet. Sie besitzen ein vergleichbares Anwendungsgebiet, da sie beide auf der MVC-Architektur basieren. Hierbei setzen sie jedoch unterschiedliche Schwerpunkte. So konzentriert sich Struts auf das Zusammenspiel zwischen den Views und dem Controller. Hierdurch ermöglicht es die leichte Integration von Benutzereingaben sowie die zentrale Konfiguration der Applikation. Durch die Möglichkeit zur Internationalisierung und dem leichten Einbinden von Fehlermeldungen können mit Struts Web-Applikationen mit sehr funktionalen Views ausgestattet werden.

Ebenfalls auf der MVC-Architektur basiert das Oracle MVC Framework, es setzt jedoch andere Schwerpunkte als Struts. So steht hier die Gestaltung des Controllers im Vordergrund, mit dem Ziel, die in ihm verwendeten Komponenten stark zu modularisieren und so ihrer Wiederverwendbarkeit zu erhöhen. Hierdurch kann der Controller oft aus schon bestehenden Komponenten zusammengesetzt werden. Da deren Aufgaben vom Framework zudem streng vorgegeben sind, erhalten die Applikationen einen ähnlichen Aufbau und können so leicht verglichen und verstanden werden. Zudem werden von Oracle MVC sowohl in den Views, als auch im Modell verschiedene Techniken unterstützt, wodurch die mit ihm erstellten Applikationen leicht in unterschiedlichen Umgebungen eingesetzt werden können.

Beiden Frameworks gemeinsam ist die Möglichkeit weite Teile der Applikation über Konfigurationsdateien festzulegen. Als Folge hiervon können verstärkt grafische Benutzeroberflächen zu ihrer Entwicklung eingesetzt werden, ein Trend der sich in den nächsten Jahren sicher noch verstärken wird. Hierdurch bewegt sich die Entwicklung von Web-Applikationen immer weiter weg von der zugrundeliegenden Technik und ermöglicht es, sich zunehmend auf die eigentliche Applikation zu konzentrieren.

Hierin liegt auch zukünftig der Nutzen bei der Verwendung von Frameworks. Gerade im Internet, in dem kontinuierlich neue Techniken, wie z.B. momentan Web-Services, eingesetzt werden können, ist es schwer sich nicht in deren jeweiligen Details zu verlieren. So sind Frameworks ein geeignetes Mittel, um bei der Entwicklung von Web-Applikationen den Überblick zu behalten und sich auf das Wesentliche konzentrieren zu können.

Literaturverzeichnis

[Bal00] BALZERT, Helmut: *Lehrbuch der Software-Technik I, Software-Entwicklung*. 2. Heidelberg, Berlin : Spektrum Akademischer Verlag, 2000

[Brä01] BRÄUTIGAM, Falko: Apache-Technologien in J2EE-Anwendungen: Cocoon und Prowler. In: *Javamagazin* (2001), 4

[BS02] BROY, Manfred ; SIEDERSLEBEN, Johann: Objektorientierte Programmierung und Softwareentwicklung - Eine kritische Einschätzung. In: *Informatik Spektrum* (2002), 2

[Ess01] ESSER, Friedrich: *Java 2 Designmuster und Zertifizierungswissen*. 1. Bonn : Galileo Computing, 2001

[GHJV96] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster*. 4. Bonn : Addison-Wesley-Longman, 1996

[Gie01a] GIESE, Dr. H.: Design Pattern and Software Architecture: V. Software Architecture, (http://www.uni-paderborn.de/cs/ag-schaefer/Lehre/Lehrver% -anstaltungen/Vorlesungen/Entwurfsmuster/WS0102/DPSA-V.pdf) / Universität Paderborn. 2001. – Forschungsbericht

[Gie01b] GIESE, Dr. H.: Design Pattern and Software Architecture: VII. Related Approaches & Tools, (http://www.uni-paderborn.de/cs/ag-schaefer/Lehre/Lehrver% -anstaltungen/Vorlesungen/Entwurfsmuster/WS0102/DPSA-VIIa.pdf) / Universität Paderborn. 2001. – Forschungsbericht

[HPRT02] HOLUBEK, Andreas ; PÖSCHMANN, Thomas ; RÖWEKAMP, Lars ; TABATT, Peter ; ROSSBACH, Peter (Hrsg.): *Tomcat 4x*. Frankfurt : Software & Support Verlag GmbH, 2002

[Leh02] LEHMANN, Mike: J2EE and Microsoft .NET / Oracle. 2002. – Forschungsbericht

[LZ02] LANGHAM, Matthew ; ZIEGELER, Carsten: XMLAnwendungen erstellen mit dem Open Source Projekt Apache Cocoon. In: *Javamagazin* (2002), 3

[Nat01] NATES, Yefim: Application Server Scenarios - From Stovepipes to Services. In: *http://www3.gartner.com/DisplayDocument?id=345860* (2001)

[Ora02] ORACLE: *Oracle 9iAS MVC Framewok for J2EE User Guide - Release 2.0 (Release Candidate)*. Oracle, 2002

[PW01] PLESSEL, Christian ; WILDE, Erik: Server-Side-Techniken im Web - ein Überblick. In: *iX* (2001), 3

[Sta02] STARKE, Gernot: *Effektive Software-Architekturen*. 1. München : Carl Hanser Verlag, 2002

[TSS01] TURAU, Volker ; SALECK, Krister ; SCHMIDT, Marc: *Java Server Pages und J2EE*. 1. Heidelberg : dpunkt.verlag, 2001

[UN02] UNDORF, Stefan ; NEUMANN, Dominik: Portalentwicklung mit Jetspeed. In: *Javamagazin* (2002), 7

Anhang A

Struts Dateien

```
<?xml version="1.0"?>

<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.0//
           EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">

<struts-config>
 <form-beans type="org.apache.struts.action.ActionFormBean">
  <form-bean name="loginForm"
          type="his.forms.login"/>
  <form-bean name="pfach"
          type="niebel.forms.fach"/>
  <form-bean name="wpfach"
          type="niebel.forms.fach"/>
  <form-bean name="pfliste"
          type="niebel.forms.fachliste"/>
  <form-bean name="wpfliste"
          type="niebel.forms.fachliste"/>
 </form-beans>
 <global-forwards type="org.apache.struts.action.ActionForward">
      <forward name="pfliste" path="/pfanzeigen.do"/>
      <forward name="wpfliste" path="/wpfanzeigen.do"/>
      <forward name="login" path="/login.jsp"/>
 </global-forwards>
 <action-mappings type="org.apache.struts.action.ActionMapping">
  <action input="/login.jsp"
        name="loginForm"
        path="/login"
        scope="request"
        type="his.actions.login"
        unknown="false"
        validate="true">
   <forward name="failure"
          path="/login.jsp"
          redirect="false"/>
   <forward name="success"
          path="/start.jsp"
          redirect="false"/>
  </action>
  <action path="/logout"
        type="his.actions.logout"
```

```
      unknown="false">
 <forward name="success"
        path="/login.jsp"
        redirect="false"/>
</action>
<action path="/wpfanzeigen"
      type="niebel.actions.wpfanzeigen"
      name="wpfliste">
 <forward name="success"
        path="/wpfanzeigen.jsp"
        redirect="true"/>

</action>
<action path="/pfanzeigen"
      type="niebel.actions.pfanzeigen">
 <forward name="success"
        path="/pfanzeigen.jsp"
        redirect="true"/>
</action>
<action path="/pfaendern"
      type="niebel.actions.pfaendern">
 <forward name="success"
        path="/pfdetails.jsp"
        redirect="true"/>
</action>
<action path="/pfspeichern"
      type="niebel.actions.pfspeichern"
      name="pfach"
      input="/pfdetails.jsp"
      validate="true">
 <forward name="failure"
        path="/pfdetails.jsp"
        redirect="false"/>
 <forward name="success"
        path="/pfanzeigen.do"
        redirect="false"/>
</action>
<action path="/wpfdetails"
      type="niebel.actions.wpfdetails"
      name="wpfach"
      input="/wpfdetails.jsp"
      validate="true">
 <forward name="success"
        path="/wpfdetails.jsp"
        redirect="false"/>
 <forward name="save"
        path="/wpfspeichern.do"
        redirect="true"/>
</action>
<action path="/wpfaendern"
      type="niebel.actions.wpfaendern"
      name="wpfach"
      input="/wpfdetails.jsp"
      validate="false">
  <forward name="wpfanzeigen"
        path="/wpfanzeigen.do"
        redirect="true"/>
  <forward name="wpfdetails"
        path="/wpfdetails.do"
```

```
        redirect="true"/>
  </action>
  <action path="/wpfneu"
        type="niebel.actions.wpfneu"
        name="wpfach">
   <forward name="success"
          path="/wpfdetails.jsp"
          redirect="true"/>
  </action>
  <action path="/wpfloeschen"
        type="niebel.actions.wpfloeschen"
        name="wpfach">
   <forward name="success"
          path="/wpfanzeigen.jsp"
          redirect="false"/>
  </action>
  <action path="/wpfspeichern"
        type="niebel.actions.wpfspeichern"
        name="wpfach"
        scope="session">
   <forward name="failure"
          path="/wpfdetails.jsp"
          redirect="false"/>
   <forward name="success"
          path="/wpfanzeigen.do"
          redirect="true"/>
  </action>
  <action path="/pool2"
        type="his.db.pool2"
        unknown="false">
   <forward name="dozent"
          path="/dozentdetails.jsp"
          redirect="false"/>
  </action>
 </action-mappings>
</struts-config>
```

Listing A.1: *Konfigurationsdatei struts-config.xml*

Anhang B

Oracle MVC Dateien

```
<?xml version="1.0" standalone="yes"?>
<!-- NOTE! Adjust the following path to match your server installation
   ... -->
<!DOCTYPE service SYSTEM "file:///z:/cora/src/dtd/Service.dtd"[
<!ENTITY docpath "">
<!ENTITY jsppath "/">
<!ENTITY xmlrulespath "file:///d:/apps/oracle/oc4j/j2ee/home/
   applications/cora/web/">
]>
<service name="Dozentenverwalten" class="oracle.cle.process.Service"
   initial="HISDekanatRouter">
   <process name="HISDekanatRouter" class="oracle.clex.process.
      ProcessRouter" end="false">
      <argument type="Integer" value="4"/>
      <argument type="Boolean" value="false"/>
      <argument type="String" value="tc1"/>
      <infoalias local="pp1" global="viewdozenten"/>
      <infoalias local="pp2" global="deletedozent"/>
      <infoalias local="pp3" global="updateDozent"/>
      <infoalias local="pp4" global="saveDozent"/>
   </process>
   <process name="deleteDozent" class="hisdekanat.process.DeleteDozent
      " end="false">
   </process>
   <process name="getDozent" class="hisdekanat.process.GetDozent" end="
      false">
   </process>
   <processgroup name="Dozenten" class="oracle.cle.process.DisplayGroup
      " end="true" initial="GetDozenten">
   <process name="DozentenRouter" class="oracle.clex.process.
      ProcessRouter" end="false">
      <argument type="Integer" value="4"/>
      <argument type="Boolean" value="true"/>
      <argument type="String" value="tc1"/>
      <infoalias local="pp1" global="viewdozenten"/>
      <infoalias local="pp2" global="deletedozent"/>
      <infoalias local="pp3" global="updateDozent"/>
      <infoalias local="pp4" global="saveDozent"/>
   </process>
   <process name="GetDozenten" class="hisdekanat.process.GetDozenten"
      end="false">
```

```
   </process>
   <process name="DozentenListePage" class="oracle.clex.process.
      CreateJSPPage" end="true">
      <argument type="String" value="/dozentliste.jsp"/>
   </process>
   <process name="DozentDetailPage" class="oracle.clex.process.
      CreateJSPPage" end="true">
      <argument type="String" value="/dozentdetail.jsp"/>
   </process>
   <process name="validateDozent" class="hisdekanat.process.
      ValidateDozent" end="false">
   </process>
   <process name="saveDozent" class="hisdekanat.process.UpdateDozent"
      end="false">
   </process>
<transition src="DozentenRouter" condition="success" dest="
   DozentenListePage"/>
<transition src="DozentenRouter" condition="failure" dest="
   DozentenListePage"/>
<transition src="DozentenRouter" condition="tc3" dest="DozentDetailPage
   "/>
<transition src="DozentenRouter" condition="tc4" dest="validateDozent
   "/>
<transition src="DozentenRouter" condition="tc1" dest="
   DozentenListePage"/>
<transition src="DozentenRouter" condition="tc2" dest="
   DozentenListePage"/>
<transition src="GetDozenten" condition="success" dest="DozentenRouter
   "/>
<transition src="GetDozenten" condition="failure" dest="DozentenRouter
   "/>
<transition src="validateDozent" condition="failure" dest="
   DozentDetailPage"/>
<transition src="validateDozent" condition="success" dest="saveDozent
   "/>
<transition src="saveDozent" condition="success" dest="GetDozenten"/>
<transition src="saveDozent" condition="failure" dest="GetDozenten"/>
   </processgroup>
<transition src="HISDekanatRouter" condition="tc1" dest="Dozenten"/>
<transition src="HISDekanatRouter" condition="tc2" dest="deleteDozent
   "/>
<transition src="HISDekanatRouter" condition="success" dest="Dozenten
   "/>
<transition src="HISDekanatRouter" condition="failure" dest="Dozenten
   "/>
<transition src="HISDekanatRouter" condition="tc3" dest="getDozent"/>
<transition src="HISDekanatRouter" condition="tc4" dest="Dozenten"/>
<transition src="deleteDozent" condition="success" dest="Dozenten"/>
<transition src="deleteDozent" condition="failure" dest="Dozenten"/>
<transition src="getDozent" condition="success" dest="Dozenten"/>
<transition src="getDozent" condition="failure" dest="Dozenten"/>
<grouptransition srcgroup="Dozenten" srcprocess="DozentenListePage"
   dest="HISDekanatRouter"/>
<grouptransition srcgroup="Dozenten" srcprocess="DozentDetailPage" dest
   ="HISDekanatRouter"/>
</service>
```

Listing B.1: *Konfigurationsdatei hisdekanat.xml*

Anhang C

Inhalt der CD

Auf der CD befinden sich die im Rahmen dieser Arbeit erstellten Web-Applikationen. Vorraussetzung für ihre Benutzung ist ein Windows Betriebssystem, sowie eine Oracle Datenbank.

Zunächst ist der Inhalt der CD auf die Festplatte zu kopieren. Die Systemvariable *java_home* ist dann auf das Verzeichnis *jdk1.3/bin* zu setzen. Das Datenbankskript ist innerhalb einer Oracle Datenbank auszuführen.

Tomcat kann über das Skript *startup.bat* gestartet werden.

Für die einzelnen Applikationen ist folgendes zu beachten:

HISDekanat:
In der Datei *web.xml* sind zunächst die Angaben zur Datenbank anzupassen. Nach dem Starten von Tomcat kann die Applikation über *http://127.0.0.1/hisdekanat* gestartet werden. Als Username und Password ist *dekanat* zu verwenden.

HISDozent:
In der Datei *web.xml* sind zunächst die Angaben zur Datenbank anzupassen. Nach dem Starten von Tomcat kann die Applikation über *http://127.0.0.1/hisdozent* gestartet werden. Als Username und Password können die Kürzel der Dozenten verwendet werden (erster Buchstabe gross).

HISDekanatoramvc:
Hier sind zunächst in der Datei *cle-providers.xml* die Datenbankangaben anzupassen. Die Applikation kann dann über *http://127.0.0.1:8080/hisdekanatoramvc/dozenten* gestartet werden.

Anhang D

Erklärung

Ich erkläre, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nichtveröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit genutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.